Comparative Analysis of the Impact of Arrival Rates on the Performance of Distance-based Streaming Outlier Detection Algorithms

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Areeha Durrani

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dr. Eleazar Leal

May 2021

Abstract

Outlier detection in data streams comes with many challenges. Among these challenges is the variable arrival rate of streams. When data packets are sent across an unreliable network, the data sending process is interrupted due to temporary loss of signal and later all of the data is tried to send at once as signals resume, resulting in data point drop, leading to faulty outlier detection. However, which algorithm performs the best in such cases remained a question until now. This research studies the impact of the arrival rate, varying queue capacity sizes, and slide sizes on the performance of state-of-the-art outlier detection algorithms for data streams. Our experiments show that using a bounded queue for incoming data points and allowing data drop has an average detrimental impact on the F-1 score, which is 100% for NETS, 99.78% for Thresh-Leap, 99.69% for Micro-Cluster, 67.5% for Exact Storm, and 0.422% for DUE. The number of outliers lost is 0% for Thresh-Leap, -0.33% for NETS, 0% for Micro-Cluster, 39.2% for Exact Storm and 38.6% for DUE, observed on default parameters.

Contents

List of Tables

List of Figures

1 Background

In this chapter, we cover the background concepts used in this thesis. In section 1.1, we talk about the motivation leading to this research. Section 1.3 talks about the research questions tackled in this work and section 1.2 talks about basic definitions necessary to understand the following discussions.

1.1 Motivation

An *outlier* is defined as "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism". [33]. This means that an outlier is a data point which behaves differently as compared to other data points generated from the source. A *data stream*, however, is an infinite collection of data points implicitly or explicitly ordered by timestamps [70]. So, the detection of outliers in data streams [2] is the problem of discovering outliers in a stream of data points. Among the issues of this problem are infiniteness, multi-dimensionality [45, 65], transience [70], concept drift [3] and uncertainty [64, 49].

The problem of outlier detection in data streams is significant [81], even more than that for the static data [42], because of its numerous applications [18, 69, 61, 73], and because data in real life is continuous and infinite, as a data stream. It helps in detecting network intrusions [26], anomalous weather patterns [59], anomalous health conditions of patients under observation, which may vary with time [28], credit card fraud detection in real-time [72], erroneous sensor readings [14, 13] and most recently

in self-driving cars [54]. Data streams in real life are uncertain [24]; for example, in contexts that require multiple sensors like distributed traffic sensor nodes talking to each other or in case of a distributed sensor network [66, 71], it can be the case that there might be a situation in which multiple sensors are connected through an unreliable network, so it is likely that some of the data points from the sensors are dropped, resulting in outliers being missed even in critical times [64]. This is critical because wireless sensor networks have numerous applications, such as in different domains for monitoring and tracking [68]. If we are unable to detect outliers in the case of an unreliable network, we cannot trust our results because this issue may occur anywhere where data is being shared through network, and then our result may be of an unknown quality [77]. As the existing distance-based outlier detection techniques [7, 60] that use distance parameters to find outliers in a system [6, 86, 43, 15, 89] have not been compared in this situation [75], therefore, our research presents an experiment to compare them on variable parameters.

Outlier detection in data streams is challenging because data points are infinitely many, so they cannot be stored [8], and also because of the uncertainty of data streams. In the case where data is taken continuously and losing even a single data point might be fatal, for example, numerous accidents in industries are caused due to sensor failures in control systems [58] or if critical sensor linked to the 2 fatal Boeing 737 Max crashes could have a way to detect anomalous behaviour or perhaps detect outlier readings, accidents could have been prevented. If the outlier detection algorithm is not efficient enough to process the data quickly, it might jeopardize the actual target of detecting outliers in real-time, for example, while taking sensor readings [79]. If the data points are lost or if the arrival rate of data points is so fast that algorithms cannot afford to lose data points, [27]. Therefore, in this research, we experimentally compare six state-of-the-art outlier detection algorithms: Exact

Storm [6], Abstract-C [86], Direct-Update-Event and Micro-Cluster [43], Thresh-Leap [15] and NETS [89] under the realistic and not uncommon assumption mentioned previously to find which algorithm performs best in those circumstances.

1.2 Definitions

In this section, we present the definitions of the terms used later in the report.

Lambda is the average rate at which the data points arrive from a source, represented as number of arrivals per second. In our case, the data points arrive at a shared queue between a data producer (from which data points arrive) and a data consumer (outlier detection algorithm). Following the approach presented in [34], the queue used here follows a *Poison arrival process*.

A *Poisson process* is a stochastic process which denote a series of discrete event, in which the average time between events is known but their exact time is random. This means that the arrival or occurrence of events are independent of each other, for example, the arrival of customers in a hospital.

The following are the properties of a Poisson process:

- Events are independent of each other.
- The average events per time period (average rate) is a constant λ.
- Two events cannot be simultaneous.

The probability density distribution for a Poisson process describes the probability of seeing n arrivals in a time period between 0 to t and is given by:

$$p_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}, \tag{1.1}$$

where t is used to define the time interval between 0 and t, n is the total number of arrivals in the interval 0 to t, *lambda* (λ) is the total average arrival rate in arrivals/sec. The term *slide* refers to the area with new or expiring data points that have been or will be a part of the sliding window, as explained in the following figure 1.1.

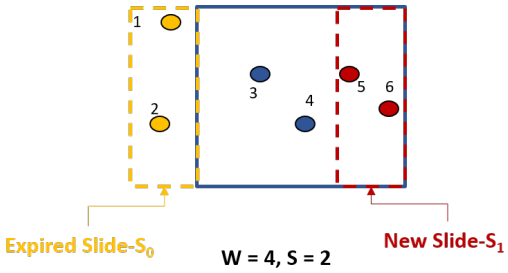

Figure 1.1: Example of Slides

The term *Concentration Ratio* originates from the field of Economics [53]. It is an indicator that shows how concentrated the data points in a dataset are, partitioned into same-sized hyper cubes, called cells. Table 1 in [89] presents the concentration ratio of different real-time datasets, where the STOCK dataset has the concentration ratio of 0.64, for TAO, it is 0.87 and for FC, it is 0.66. In order to better understand the concept, consider the examples presented in figures 1.2 and 1.3. Here, the figure 1.2 shows the example of a dataset with low concentration ratio and the figure 1.3 shows the example of a dataset with slightly higher concentration ratio.

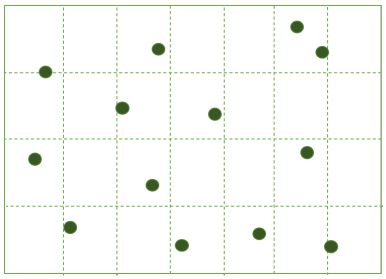

Figure 1.2: Example of dataset with Low Concentration Ratio

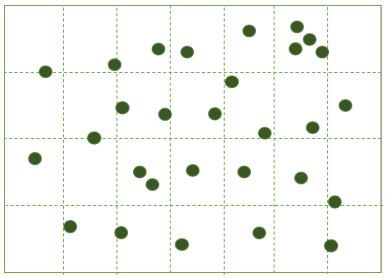

Figure 1.3: Example of dataset with High Concentration Ratio

It can be calculated as shown in equation 1.2:

$$Concentration\ Ratio = \frac{Total\ data\ points\ in\ the\ top\ quarter\ of\ most\ populated\ cells}{Data\ points\ in\ the\ entire\ data\ space} \tag{1.2}$$

In case of high concentration ratio, the data points in the same slide are close to one another, which is referred to as *Intra-Slide Proximity*, as described in [89] and used in several other fields [55]. In case of high concentration ratio, the data points in the expired slide are close to data points in the new slide, which is referred to as *Inter-Slide Proximity*, as described in [89].

In order to observe the effect of arrival rates on the performance of outlier detection algorithms, the *producer-consumer technique* [36] is used, which enables the data stream generation program and the detection algorithm to share a blocking queue, as in [82]. The data points are continuously fed into the queue by the producer, while the detection algorithm (consumer), takes them one-by-one as it processes previous points. The queue has a fixed capacity set to the default value. If at any point the data points from the producer find the queue to be full, they are dropped instantaneously [4]. Therefore, the quicker the algorithm takes the points, the lesser it loses them and thus detects the outliers more accurately. It is observed that by increasing the arrival

rate (lambda), the inter-arrival rate between the data points decreases, resulting in data points rushing quicker towards the queue, which in turn impacts on the total number of outliers detected.

As these infinite, high-volume and continuous data streams cannot be stored, in order to process the queries on them in real-time, the *sliding window* model [9, 63] is used to restrict the range of continuous queries [31], which may be count-based or time-based depending upon the choice of the type. If the restricted items are up to N most recent items, it is called a *Count-based* sliding window model [90], while if the timestamps of the items are as old as current timestamp minus T, it is called a *Time-based* sliding window model [17, 50].

1.3 Research Questions

The research questions studied in this thesis are the following:

1. What is the overall impact of the arrival rate of data points on the performance of the outlier detection techniques for data streams?

2. In the case of high arrival rates, the shared queue between data producer and consumer gets full and cannot accept any more data points, for example, in case of an unreliable network, the data points might be dropped or lost. What percentage of outliers do the different outlier detection algorithms fail to detect?

1.4 Contributions

The contributions made by this research in the scientific community are as follows:

- We present the first experimental study comparing the existing outlier detection

algorithms on the basis of arrival rate. We study the impact of arrival rate on the total execution time, average waiting time, total number of outliers lost, precision, recall and f-1 score.

- We perform an experimental comparison of outlier detection algorithms on the impact of queue capacity and slide size. We found that when the queue capacity is small, the average waiting and total execution time is small, while if it increases, both remain constant.

- Our experiments show that the average waiting time for Exact Storm and DUE algorithms decreases as they drop more data points with the increase in slide size, while those who do not drop data points, observe an increase in the average waiting time.

- We found that the performance of outlier detection algorithms for data streams is dependent on the arrival rate (lambda) of the data points, for all algorithms. We found that the average impact of arrival rate on the number of outliers detected for each algorithm is: 39.2% for Exact Storm, 38.6% for DUE, 0% for Thresh-Leap for NETS, 0% for Micro-Cluster and -0.33%.

- We found through our experiments that using a bounded queue for incoming data points and allowing data drop, arrival rate has a significant detrimental impact on the F-1 score for all algorithms, across all datasets. We found that the average impact of arrival rate on F-1 score for each algorithm is: 99.78% for Thresh-Leap, 100% for NETS, 99.69% for Micro-Cluster, 67.5% for Exact Storm and 0.422% for DUE.

2 Related Work

In this section, we explain the existing distance-based outlier detection algorithms used in [76]: Exact-Storm, Abstract-C, Direct-Update-Event, Micro-Cluster, Thresh-Leap algorithms and NETS.

2.1 Exact Storm

The *Exact Storm* algorithm [6] uses the index structure shown in figure 2.1 to store data points for each window. For each data point o in the index structure, its *preceding neighbors*, which are defined as the data points that are the neighbors of the data point o and expire before o, are stored in the list *o.pn*, and its *succeeding neighbors*, which are defined defined as the data points which are neighbors of the data point o and expire in the same slide or after o, are stored in the list *o.sn*.

When the window slides and data points expire, they are removed from the index structure but not from the *o.pn* list of the data points after that, as shown in case of data point o_3 in window W_1, in figure 2.1. It can be seen that in window W_1, although data points o_1 and o_2 are removed from the index structure, they are written as the preceding neighbor of data point o_3. In case of a new slide, a range query returns the neighbors in range R, which in this case results in finding o.pn and o.sn for data points o_7 and o_8.

If a data point o has less than K preceding and succeeding neighbors, in case of given example: 2 (assuming $K = 2$), then it is an outlier. In the given example, data

point o_6 could be an outlier since it has fewer than K neighbors.

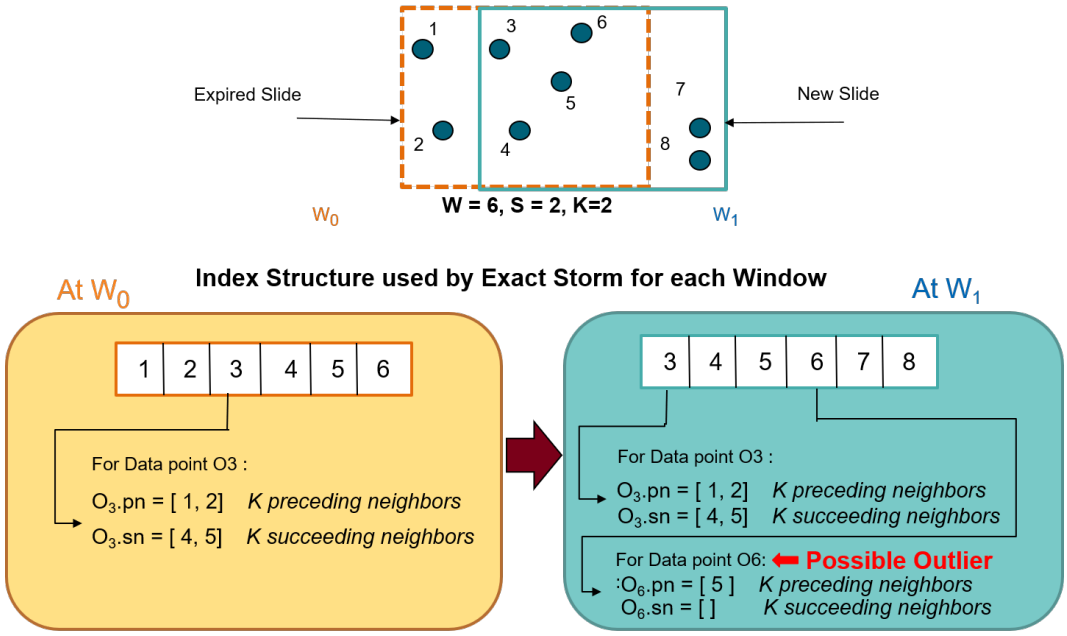

Figure 2.1: Example of Exact Storm

For the Exact Storm algorithm, an outlier is a data point whose sum of preceding and succeeding neighbors is less than K. In other words, when the following statement holds $(o.sn + o.pn) < K$.

The index list stores preceding neighbors for a data point but calculates both preceding and succeeding neighbors, which makes it less optimal in terms of memory usage. Moreover, since expired preceding neighbors are not removed from the list, it takes extra time to retrieve preceding neighbors for active data points, it requires extra CPU time to perform the operation.

2.2 Abstract-C

Abstract-C [86] stores the count of neighbors for every data point o in every window it participates in, which is called *ln_cnt*. It also uses the index structure, like

Exact Storm as shown in figure 2.1. The intuition behind Abstract-C is that each point participates in W/S windows, which is the maximum size that the sequence $o.ln\_cnt$ can go up to, for each data point o. For understanding, consider the example presented in figure 2.2. Here, data point o_3 is considered. As described earlier, the size of $o_3.ln\_cnt$ can max be 3 as W/S, which in this case is 3. This means that it can participate in 3 windows at max.

In case of window W_1, it has o_1 and o_2 as neighbors, leading to $o_3.ln\_cnt$ for W_1 be [1,1,0] . o_1 is no more a neighbor in W_2, in fact there is a new neighbor o_4, leading to $o_3.ln\_cnt$ to be [2,1]. In case of W_3, o_2 is no more a neighbor but there is a new neighbor o_5 and old neighbor o_4, leading to $o_3.ln\_cnt$ be [2].

The outlier for Abstract-C is the data point o if $o.ln\_cnt[0]$ is less than K, which means o_3 in the given example is not an outlier, as it has neighbors greater than K, as shown in the figure 2.2.

For the Abstract-C algorithm, an outlier is a data point whose $o.ln_cnt[0]$ has neighbors less than K. In other words, when the following statement holds $o.ln\_cnt[0] < K$.

This setting makes it not at all suitable in terms of memory, as memory requirement depends upon the input data stream. This means that it is not suitable for small slide size S, as the size of $o.ln\_cnt$ is equal to W/S, resulting in W/S lists for each data point.On the other hand, it does not spend time in finding active preceding neighbors for each data point and therefore takes less time as compared to Exact Storm.

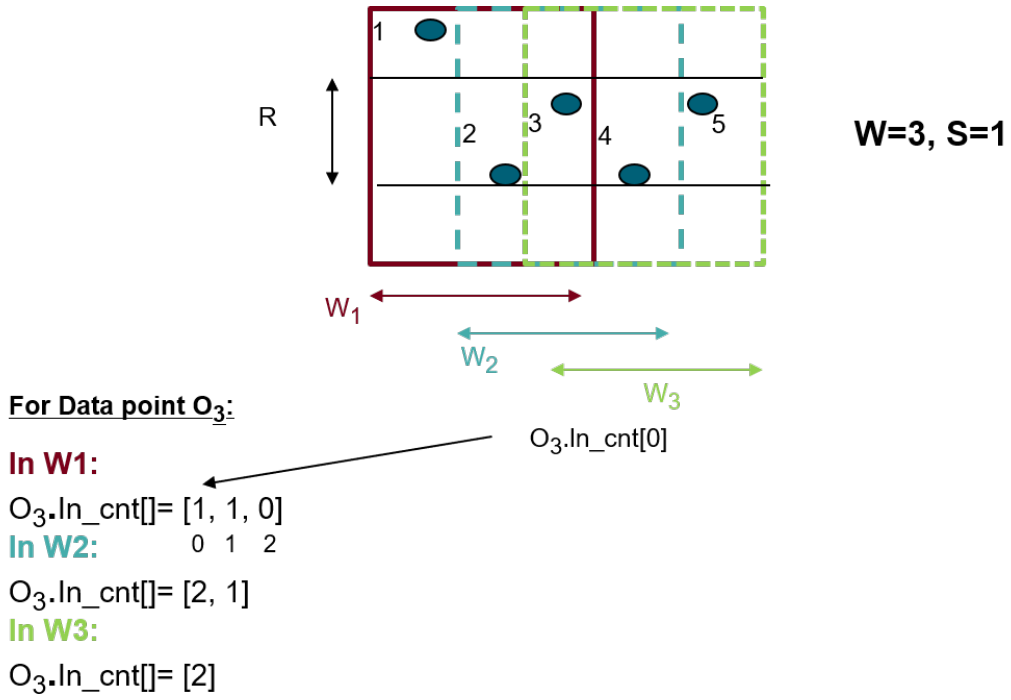

Figure 2.2: Example of Abstract-C

2.3 Direct-Update-Event (DUE)

Direct-Update-Event (DUE) [43] also uses an index structure, like Exact Storm and Abstract-C, as shown in the figure 2.1, and it relies on the idea that as a result of the sliding window, only those data points which are neighbors to the expired data points, are updated in the structure. A priority queue-*event queue* stores all unsafe inliers, which sorts them in the order of increasing expiration time of their preceding neighbors while the *outlier list* stores all the outliers for each window.

Figure 2.3 shows an example of how DUE works. As the window slides from W_0 to W_1, the expired data points ($o1$ and $o2$) are removed from the index structure and the event queue is checked for updating the neighbors list of those unsafe inliers who were neighbors to these expired data points. If any of these unsafe inliers do not have

the neighbors greater than or equal to K, it is moved to the Outlier list. On the other hand, data points can be moved from Outlier list to the event queue if more of its succeeding neighbors are discovered in the new window, summing up its total neighbors to be greater than or equal to K, thus making it a safe inlier.

For Direct-Update-Event algorithm, at the end, the *Data points in Outlier List* are reported to be outliers.

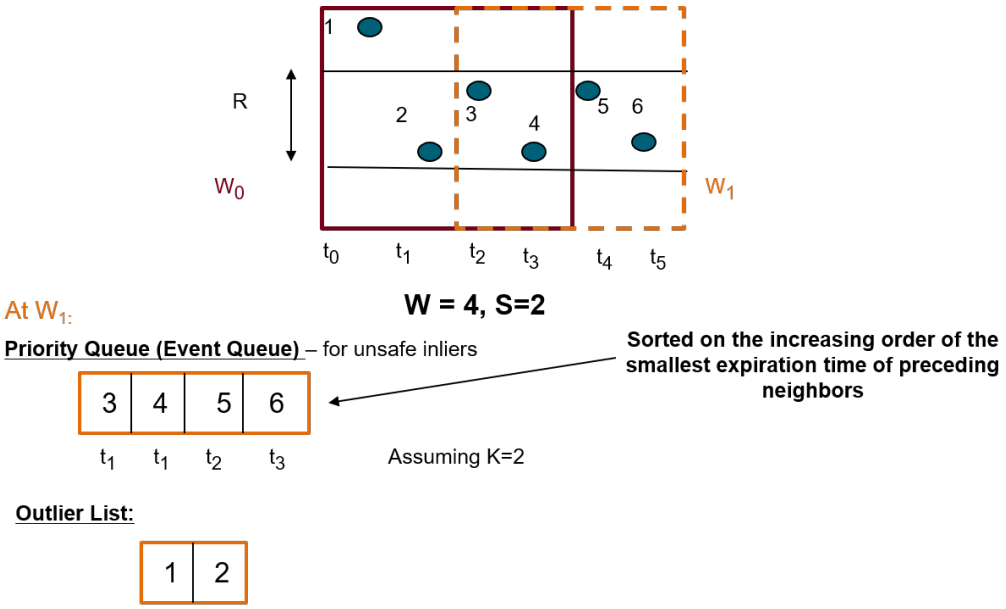

Figure 2.3: Example of DUE

This algorithm is efficient because whenever the window slides, it re-evaluates only those data points which are closely affected by the expiring data points but it requires extra time and memory to maintain and sort the priority queue.

2.4 Micro-Cluster-Based Algorithm

Micro-Cluster-based algorithm [43] introduced the idea of storing neighboring data points in micro-clusters, eliminating the need of expensive range queries, which grew

more expensive with the size of the dataset.

Each micro-cluster centers around one data point, just like K-nearest neighbors [44]. This is why the size of each micro-cluster is $K + 1$. Other data points within the same cluster must be in the range of $R/2$ to be a part of the cluster, where R represents the threshold distance, set manually by us, depending upon the dataset. Each data point inside a cluster is therefore an inlier. However, as the window slides, the expired data points are removed from micro-clusters. In such case, a micro-cluster has less than $K+1$ data points, it is dispersed and the data points from these micro-clusters are dealt as if they are new data points, only if they do not expire with the window.

Those data points that do not have neighbors greater than or equal to K, are not a part of any micro-cluster, unlike MC1, MC2 and MC3 in the figure 2.4. These may be outliers or inliers as they can be neighbors with data points from different clusters and are therefore stored in a separate list *PD*. Unsafe inliers are those data points that are not a part of any cluster and are stored in a list, which uses an event queue, like DUE. However, as the window slides, the expired data points are removed from PD.

For Micro-Cluster algorithm, at the end, the *Data points in PD List* are reported as outliers.

Since a micro-cluster represents the neighborhood information for each data point in its cluster. Just as each leader of the group can represent the group, there is no need store information of each member of the group, this algorithm takes less memory. Moreover, it takes less time because it saves time for neighbor search, as the micro-cluster data points are arranged on the basis of distance.

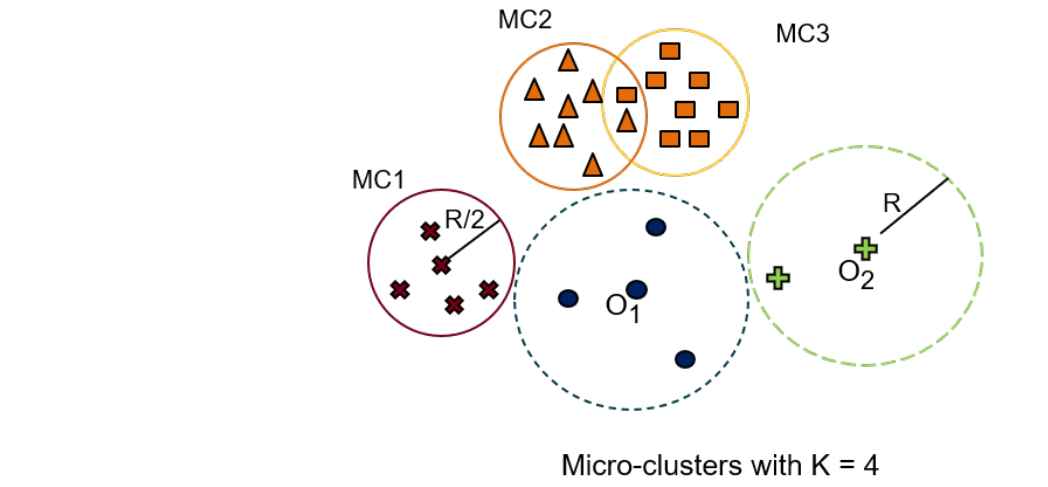

Figure 2.4: Example of Micro-Cluster

2.5 Thresh-Leap

Thresh-Leap [15] eliminates the need of complete neighborhood search for range query, by introducing the idea of minimal probing principle. The intuition behind this principle is that detecting outliers is basically separating outliers from inliers. This means that for a data point O, the distance is calculated between it and other data points in the window, until the area around it (within set threshold R) is evaluated for inliers. If it still does not have neighbors greater than or equal to K, it is considered an outlier.

Firstly, for a data point, the algorithm searches for the succeeding neighbors first and then the preceding neighbors in the reverse chronological order, in order to look for K nearest neighbors. For each data point O, the number of neighbors of O in every slide are stored in a list - $O.evil[]$ and a *Trigger List* stores the data points whose outlier status is unknown and can be affected as the slide expires.

For Thresh-Leap algorithm, an outlier is a data point whose sum of preceding neighbors in the $o.evil[]$ list and succeeding neighbors is less than K.In other words, when the following statement holds $(o.pnino.evil[]) + (o.sn[]) < K$.

As a result, there exists a small index structure per slide to carry out range queries.As the slide expires, this index structure is discarded and the data points in the Trigger list are re-evaluated. The corresponding data points in $O.evil[]$ are removed and if it has less than K neighbors, the succeeding slides never probed before are probed. This re-probing for the expiring slides $S1$ and $S2$ can be seen in the figure 2.5.

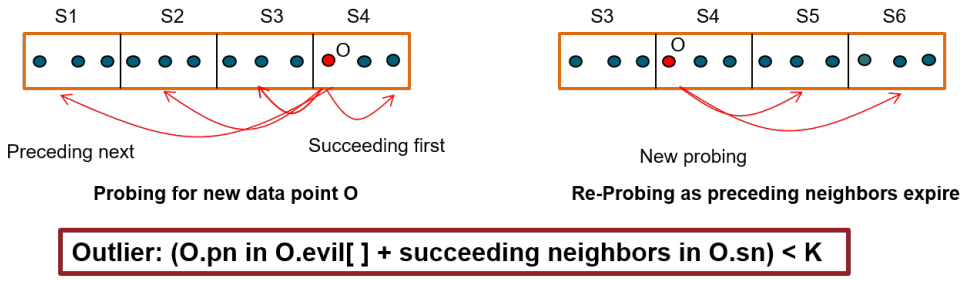

Figure 2.5: Example of Thresh-Leap

Thresh-Leap takes less time as compared to Exact Storm, Abstract-C and DUE as it saves time in looking around the neighborhood of the data point only, rather than all the slides. However, it is inefficient for memory at lower slide size because the smaller the slide size, the more the slides and thus more probing.

2.6 NETS

NETS [89] emphasizes on the concept of concentration ratio, which implies that the data points in a data stream are actually concentrated in the form of sets in the regions of the data space and therefore, introduces the idea of set-based update approach for distance-based outlier detection, utilizing not only the inter-slide proximity but also proposing the idea of *intra-slide proximity*, which is discussed in section 1.2. This means that other algorithms process the expiring and the new slides separately one-by-one, while NETS processes them both together, in a concurrent manner.

The data points that are close to each other, are grouped together into a set, to avoid repetitive updates for individual data points, occurring as a result of a sliding window. This allows it to observe the net change between expired and new data points in each set, eliminating repetition.

This set-based approach first compares the close data points in the expired and new slide and manages them concurrently to calculate the net effect for each set, which is $\pm$ 0 if there is no net change, +1 if there is a net change of new data point and -1 if there is a net change of expired data point, as shown in the figure 2.6.

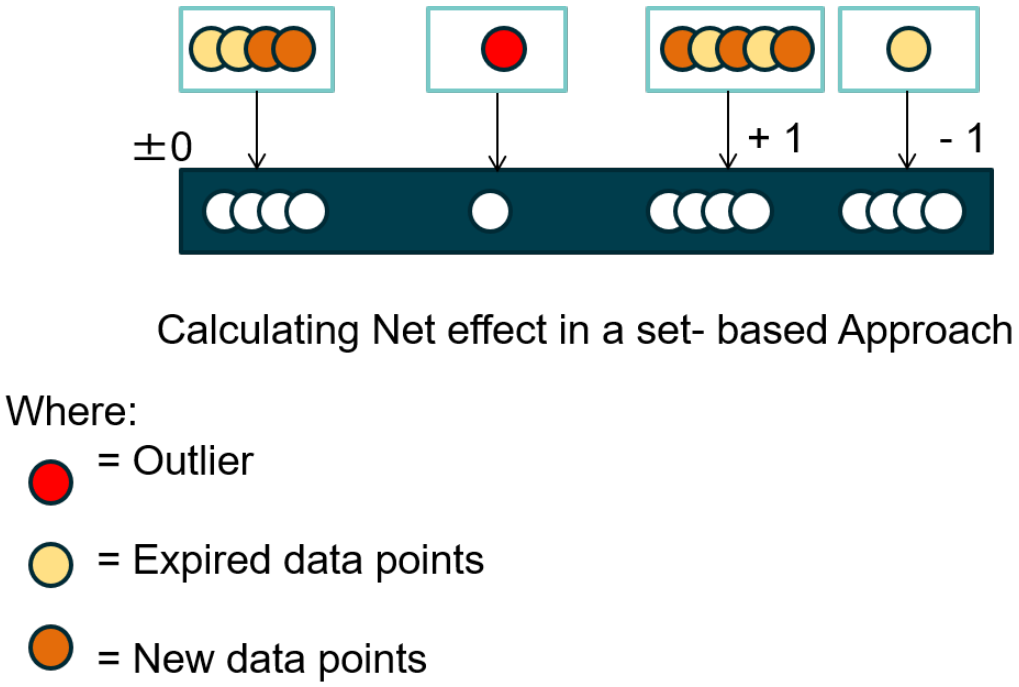

Figure 2.6: Example of NETS

This NETS based technique requires fewer updates and taking the net effect reduces the time and unnecessary labeling of a data point an outlier and then renaming it an inlier after finding neighbors greater than or equal to K.

3 Experiments

In this chapter, we discuss the experimental setup and the default parameters in section 3.1, datasets used in section 3.2, hardware used in section 3.3, parameters used to measure performance in section 3.4, experimental results in section 3.5 and final results comparing the performance of algorithms using precision, recall and F-1 score in section 3.6.

3.1 Experimental setup

We model the data stream point arrival with a Poisson distribution [20, 40] with parameter λ, the arrival rate of points, and following the Producer-Consumer logic [39], in which the data producer continuously produces data and sends it to the consumer through a shared queue. The consumer continues to take the data and process it according to its speed to send and receive data. The queue has a fixed capacity set to the default value of 500, except while observing the impact of queue capacity on the performance of outlier detection in section 3.5.3. If at any moment, the queue is full, it does not take any more data points and each of the data points coming from the data stream producer is then dropped. This can be explained in figure 3.1.

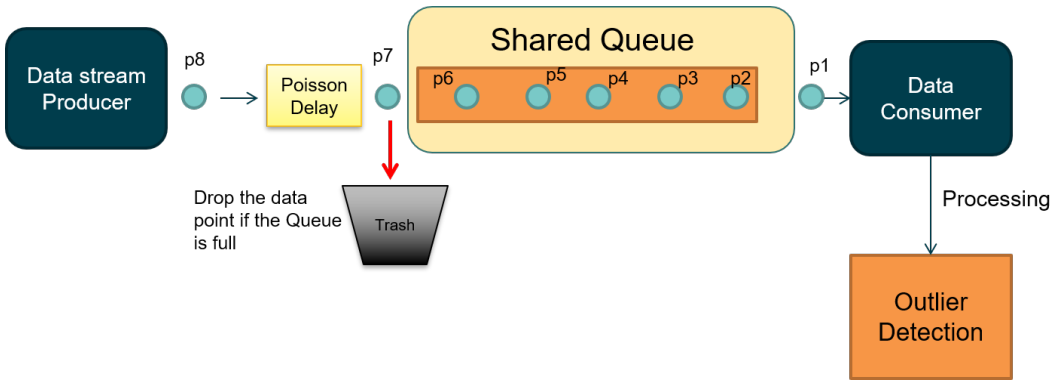

Figure 3.1: Producer-Consumer Logic

As data points arrive, the *sliding window* technique [30] is used to observe the active data points. Out of the two sliding window techniques: *Time-based* [29], in which the number of windows are specified in terms of their time stamps and *Count-based* [90], in which the number of windows are specified in terms of their count. These algorithms use the count-based sliding window approach [21, 25] to detect outliers.

Default values for each parameter vary, depending upon the experiments and dataset under observation. Here, *Window Size* W represents the number of data points considered in a sliding window, *Number of windows* represents the number of windows to consider for experiments, *Slide Size* S represents the number of new data points being taken and expiring at the same time in a window, *Lambda* λ represents the number of data points or stream of slides per second and is the mean value of the exponential distribution for the stream generating process, *Queue Capacity* Q, as used in [87], represents the capacity of the blocking queue receiving the data, K represents the number of nearest neighbors to the data point and *Distance Threshold* R represents the minimum distance required to be considered as a neighbor to a data point.

Consider the figure 3.2 to understand the above mentioned parameters. In this

figure, *Lambda* is set to 3, Q to 4, W to 4, S to 2, *Number of Windows* to 2 and the outliers are represented in red if K is set to 2.

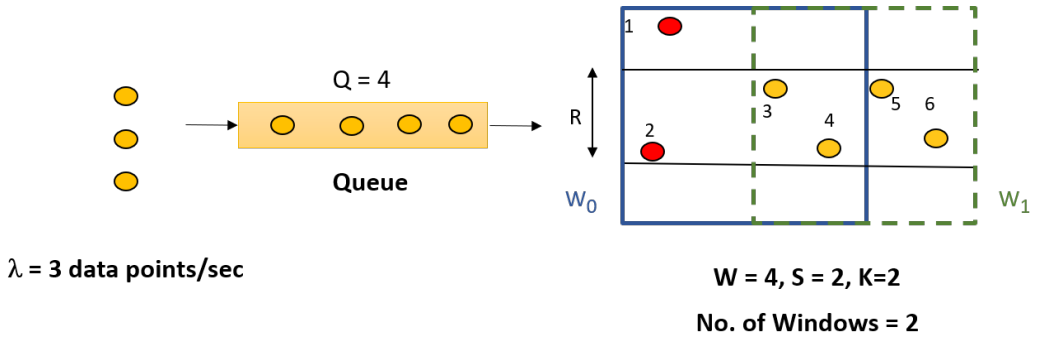

Figure 3.2: Example of Parameters

All the fixed parameters are set to the same values as were used in the previous experiments [6], [86], [43], [15], [89]: For example, for the dataset Forest Cover, the R is set to 525, 1.9 for TAO, 0.45 for STOCK, as were used in the referenced paper [76]. Table 3.1 presents the default values used for experiments.

Parameter	Range	Default
Window Size	100 – 20K	5K
Number of Windows	100 – 20K	10K
Slide Size	1 – 20K	500
Arrival Rate (Lambda)	100 – 100K	10K
Queue Capacity	1 – 10K	500

Table 3.1: Default Parameters Used in Our Experiments

3.2 Datasets

The following datasets were used in the paper [76] and are used in our experiments:

1. **Forest Cover (FC)** [1] contains 581,012 records with 55 attributes-available at UCI KDD archive. The dataset is used for predicting forest cover type and is a classification problem with various continuous and categorical geographic measurements.

2. **TAO** [2] contains 575,648 records with 3 attributes-available at Tropical Atmosphere Ocean project. This dataset displays moored ocean buoys. Information collected includes subsurface, sea surface and air temperature, ocean salinity, wind speed and direction, and short and long wavelength solar radiation data.

3. **Stock** [3] contains 1,048,575 records with 1 attribute-available at UPenn Wharton Research Data Services. This data represents stock prices and was collected by Wharton Research services from individual resources of various companies.

3.3 Hardware

The experiments were performed on the *Janus machine* at the University of Minnesota Duluth, with x86-64 architecture, 1200 MHz CPU, 2 threads per core with 8 cores per socket and a total of 2 sockets.

3.4 Performance Measures

The performance measures used in the experiments are Total Execution Time, Average Waiting Time, Precision, Recall and F-1 Score, as were used in [32].

Total Execution time is the total time elapsed for all the data points to reach the consumer and get processed. *Average Waiting time*, as used in [5] [41], is the

[1] https://kdd.ics.uci.edu/
[2] http://www.pmel.noaa.gov
[3] https://wrds-web.wharton.upenn.edu/wrds/

time each data point has to wait in the queue before it is taken by the Consumer for processing. *Precision* is used to measure the amount of the data points that are outliers, out of the predicted values. Precision can be calculated as shown in the equation 3.1:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.1}$$

Recall is used to measure the amount of data points that are outliers out of the actual values. Recall can be calculated as shown in the Equation 3.2:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.2}$$

F-1 score is used to validate the accuracy of the detection by algorithms. F-1 score can be calculated as shown in the Equation 3.3:

$$\text{F-1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

3.5 Experimental Results

In this section, we evaluate the impact of arrival rate, queue capacity and slide size on total execution time, wait time and the outliers detected vs actual outliers.

3.5.1 Impact of Arrival Rate (Lambda) on Total Time

As mentioned in section 1.2, Lambda is the average arrival rate of data points measured in number of point arrivals per second [80]. As lambda increases, the inter-arrival time between the data points decreases. This means that the data point, come

quicker one after another. As a result, the total execution time decreases with the increase in lambda. When Lambda = 1000, all the algorithms have similar execution times, except NETS because the data points are arriving so slowly that most of the execution time is spent waiting for the data points. This means that even if the algorithms differ in performance, their difference in performance is negligible in the case of the FC and TAO datasets in Figure 3.3, Figure 3.4. However, that difference can be observed in the case of the STOCK dataset, in Figure 3.5, which is twice as big as the others. For the FC and TAO datasets, NETS take a constant Execution time while for STOCK, it takes almost 10 times less time as compared to the other algorithms.

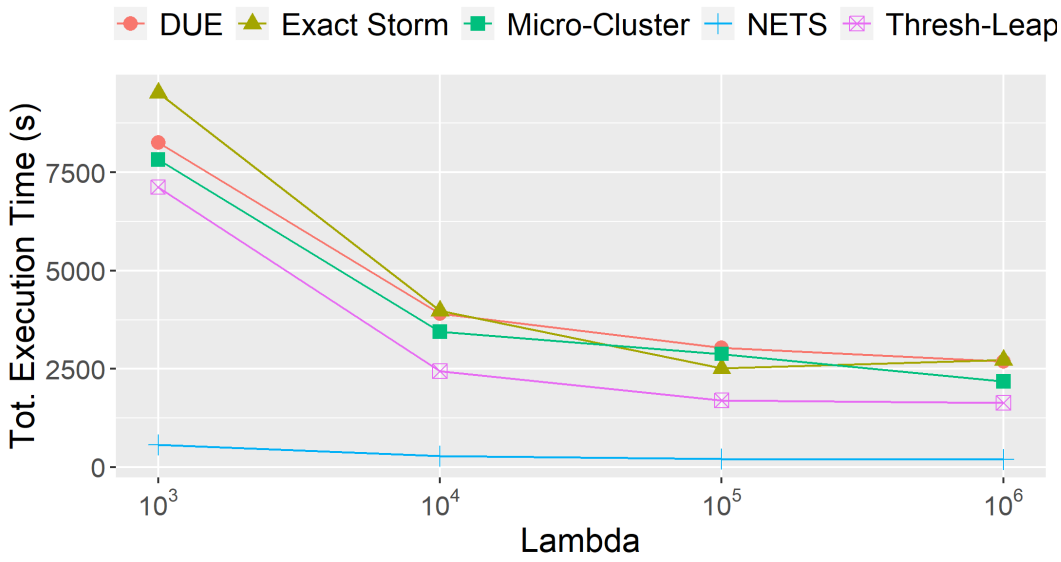

Figure 3.3: Impact of Lambda vs Total Execution Time for the FC dataset

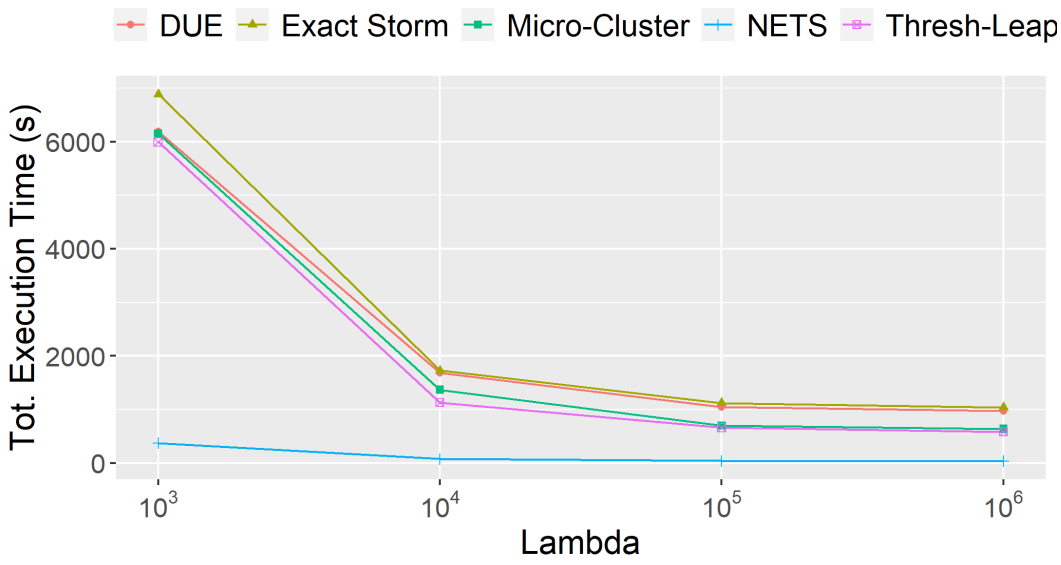

Figure 3.4: Impact of Lambda vs Total Execution Time for the TAO dataset

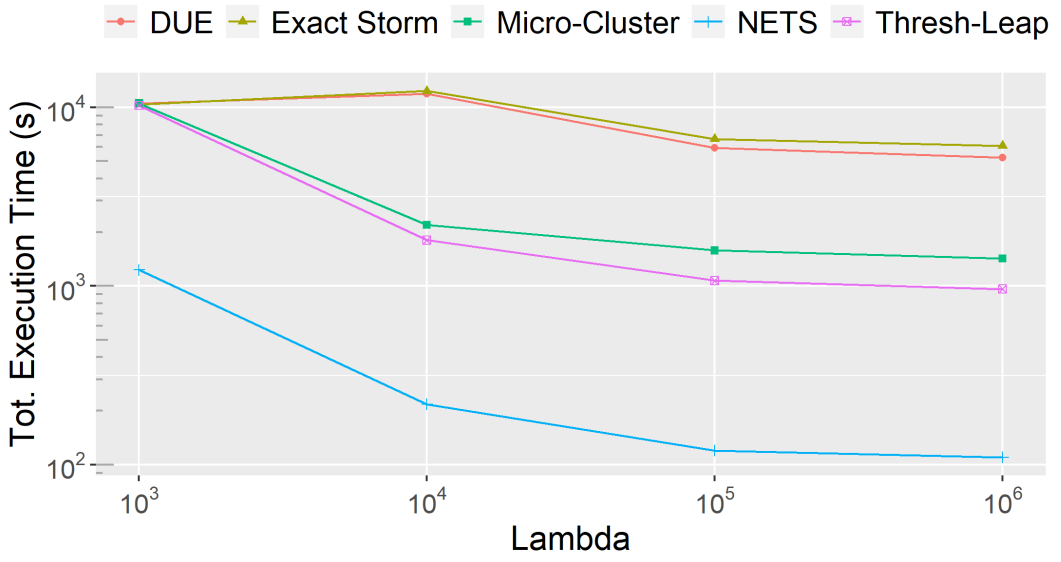

Figure 3.5: Impact of Lambda vs Total Execution Time for the STOCK dataset

The average time that the data points have to wait in the queue to be taken by the Thresh-Leap and Micro-Cluster algorithms is less as compared to other algorithms,

which can be observed in Figures 3.6, 3.7 and 3.8.

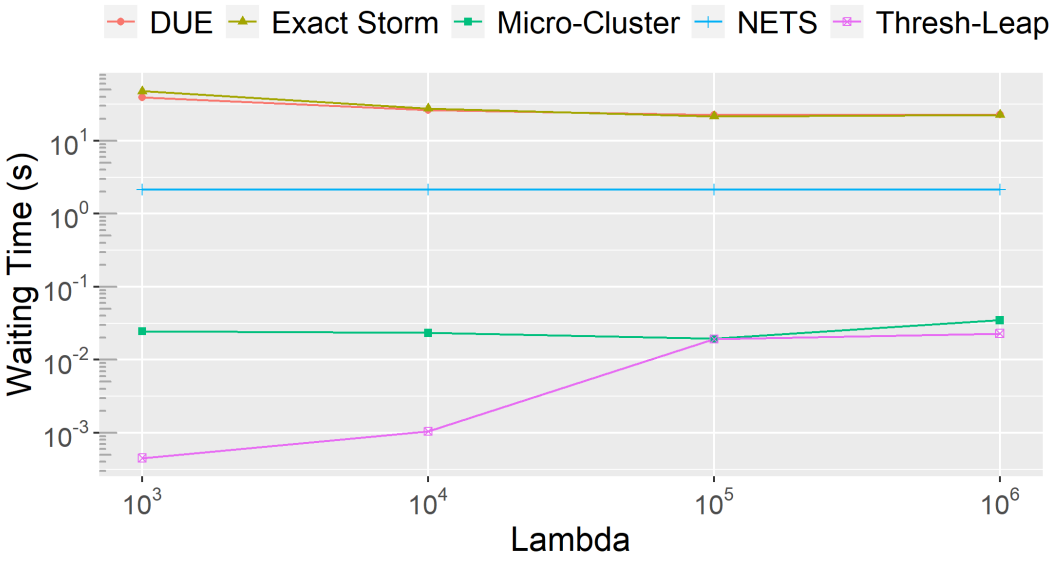

Figure 3.6: Impact of Lambda vs Average Waiting Time for the FC dataset

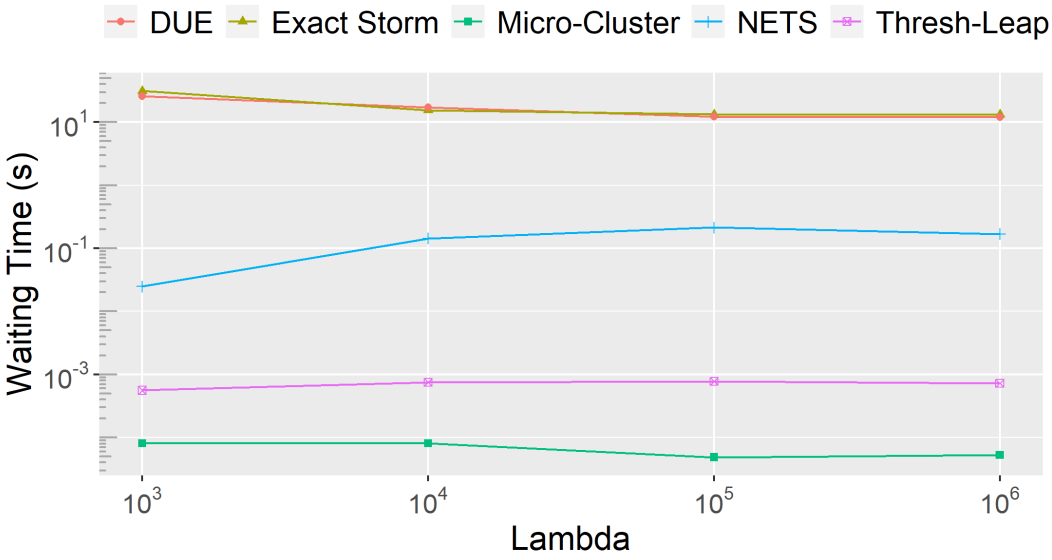

Figure 3.7: Impact of Lambda vs Average Waiting Time for the TAO dataset

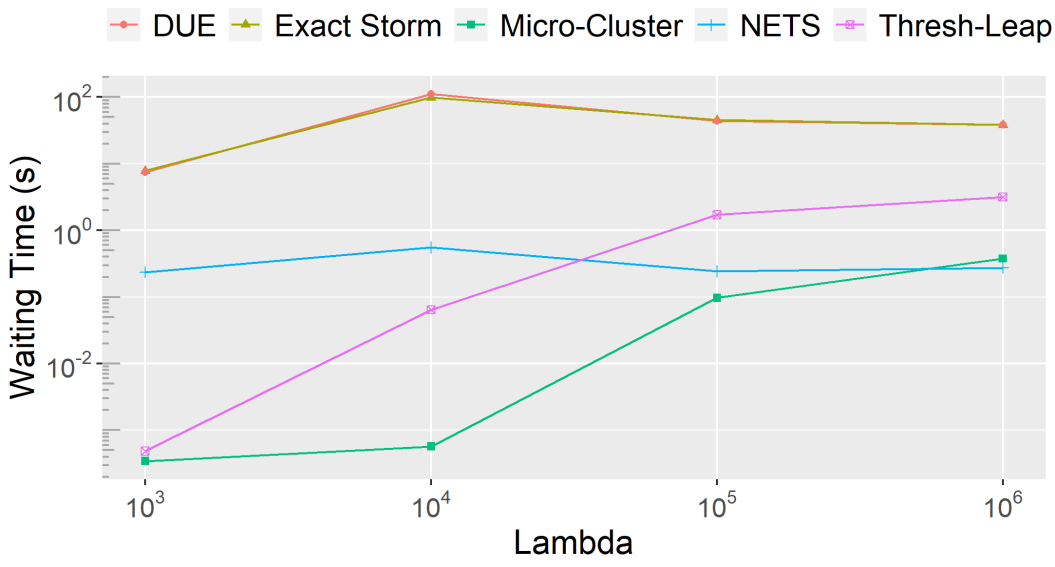

Figure 3.8: Impact of Lambda vs Average Waiting Time for the STOCK dataset

As the data points arrive to the queue, the latency with which they are taken by each algorithm may vary depending upon how fast an algorithm works. As Lambda increases, the data points rush even faster towards the queue, but when the queue is full, they are dropped out. The sooner the algorithm takes them for processing, the lesser the number of data points are dropped. As a result, if an algorithm loses points, it detects fewer outliers as compared to the actual amount before when the queue has infinitely large capacity. This can be observed in Figures 3.9, 3.10 and 3.11. *Abstract-C, however, cannot be observed in this scope because it cannot afford to lose data points, if it does lose data points, it cannot detect the subsequent data points.* The reason of this behavior is that Abstract-C relies on the rank of the data points to update the neighbors of an object, so if some data points are lost in between, the algorithm fails to perform the same way. Therefore, it is not possible to study the effect of varying the arrival rate (lambda) on it. The novelty of each algorithm can be found by comparing the total number of outliers detected by the algorithm

at the end of the experiment. It can be observed that some algorithms lose data points as lambda increases and thus, their results are not complete, while others do not lose data points, no matter how much lambda increases. These experiments are observed considering the results produced by actual algorithms, also referred in [76] as the *ground truth*, which means that the total number of outliers detected by each algorithm during this experiment are compared with the total number of outliers detected by the actual algorithm in [76], observed without varying the arrival rate.

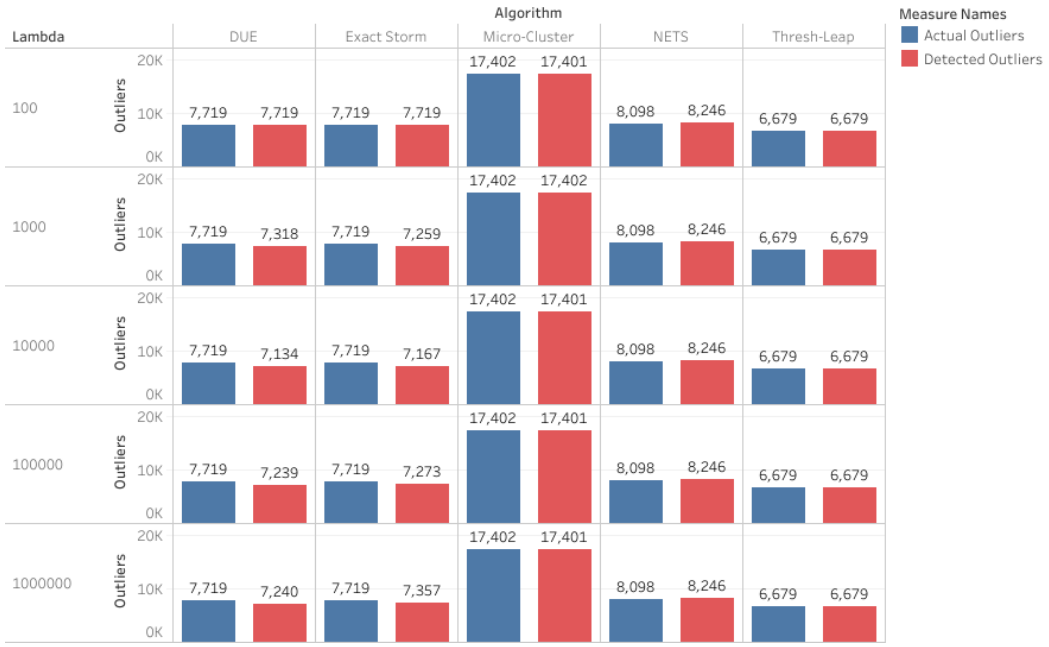

Figure 3.9: Lambda vs Total Number of Outliers Detected for the FC dataset

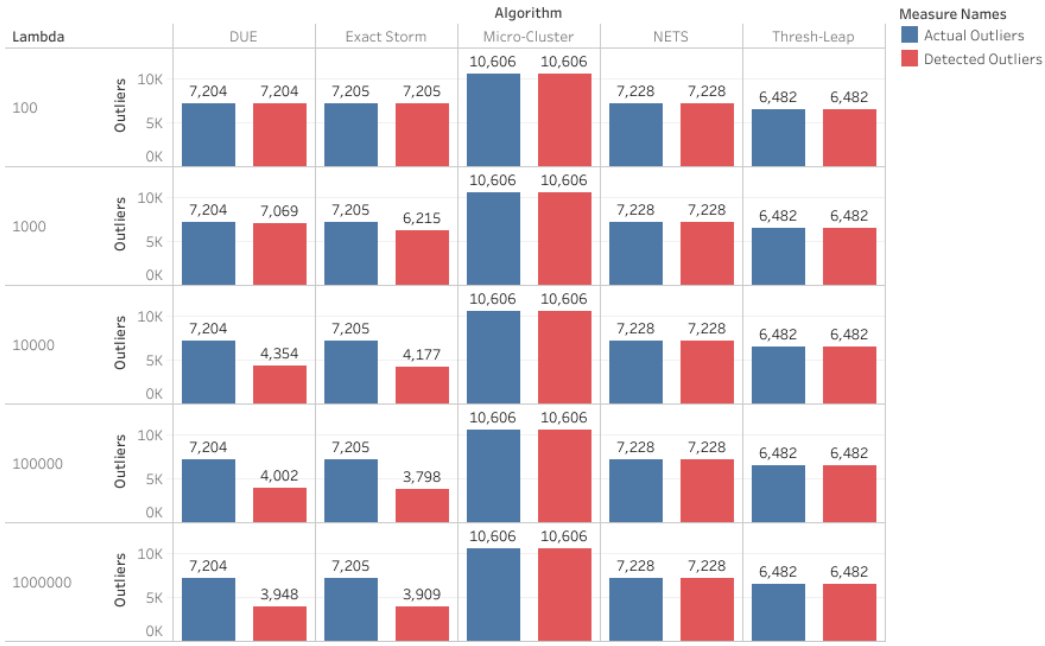

Figure 3.10: Lambda vs Total Number of Outliers Detected for the TAO dataset

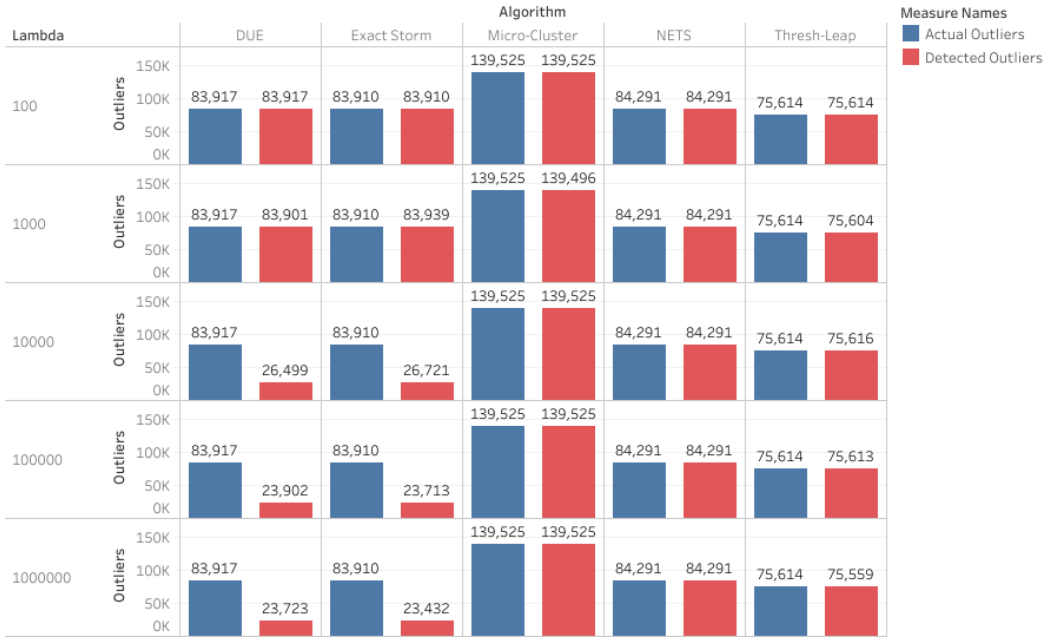

Figure 3.11: Lambda vs Total Number of Outliers Detected for the STOCK dataset

It can be observed in Figures 3.6, 3.7 and 3.8 that as lambda increases, the average

waiting time for DUE and Exact Storm becomes constant. This is because these algorithms fail to process the data points fast enough to not let the queue be full, which results in them losing data points and detecting fewer outliers as compared to actual values, as can be seen in Figure 3.9, Figure 3.10, Figure 3.11.Those data points which were supposed to wait for their turn are now dropped when they find the queue full. Whereas, for Thresh-Leap, Micro-Cluster and NETS, the waiting time slightly increases or stays constant as both the algorithms continue to not miss any data point for outlier detection, not letting the queue to be filled. It can be seen in the experiments that NETS consumes less Total execution time as compared to other algorithms but has greater average waiting time as compared to Thresh-Leap and Micro-Cluster algorithms, which means most of its time is spent waiting for data points in the queue.

3.5.2 Discussion

- As lambda increases, the inter-arrival time between data points decreases, resulting in the data points rushing towards the queue and decreasing the total execution time.

- Exact Storm and DUE algorithms lose data points and thus maintain the average waiting time.

- The impact of arrival rate on the number of outliers lost (at the default parameters) is 0% for Thresh-Leap, -0.33% for NETS, 0% for Micro-Cluster, 39.2% for Exact Storm and 38.6% for DUE.

3.5.3 Impact of Queue Capacity on Total Time

In this experiment, instead of assuming a queue of infinite capacity, we study the impact of the size of the queue on the total execution time and average waiting time of the algorithm, as the queue capacities have a direct impact on the latency of data [47] and numerical examples in [22] indicate that there might be an impact of the finite queue capacity on system performance.

If the data points are made to wait till the space is available in the queue or till it is full, increasing the queue capacity decreases the average waiting time because the chances of a data point to wait till the queue is full decreases, as the queue can accommodate more data points in this case. On the other hand, if the data points are dropped if the queue is full (the case we follow), if data points are made to wait till the space is available in the queue or till it is full, when the queue capacity is increased, more data points can be accommodated inside the queue at once, thus maintaining the average waiting time of the data points. Therefore, the total execution time also remains constant, as can be observed in Figure 3.12, 3.13 and 3.14.

It can be observed in the figures that for the Exact Storm and DUE algorithms, when the queue capacity is small, the average waiting times for the algorithms are low because the data points did not have to wait in the beginning, rather, seeing the queue full were dropped.

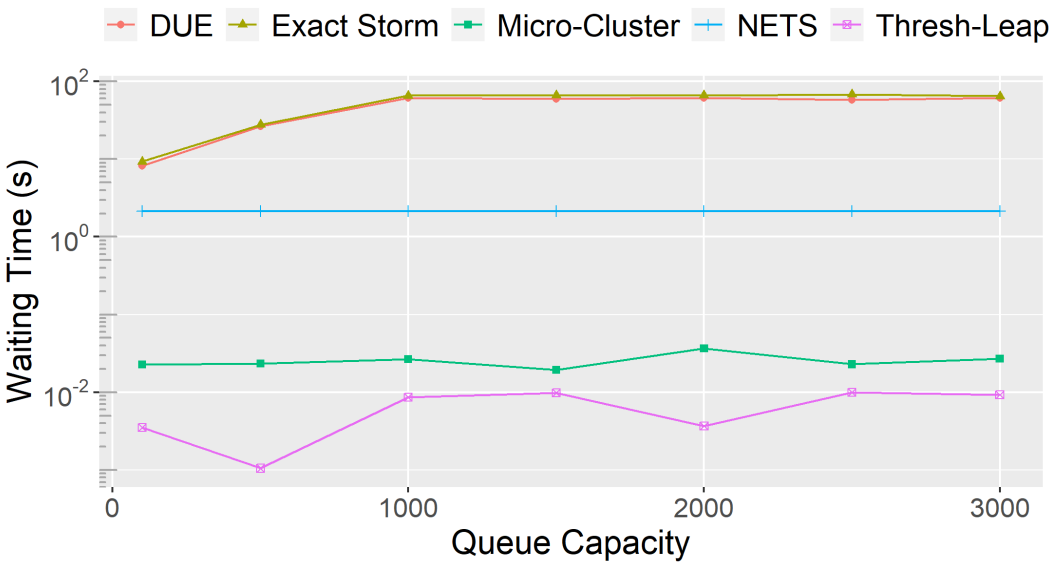

Figure 3.12: Impact of Queue Capacity vs Average Waiting Time for the FC dataset

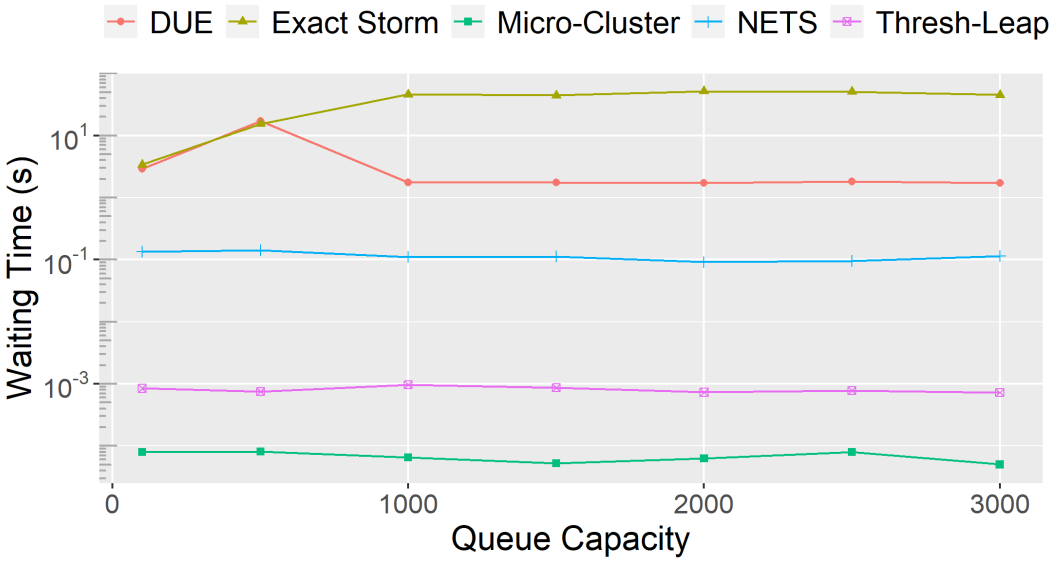

Figure 3.13: Impact of Queue Capacity vs Average Waiting Time for the TAO dataset

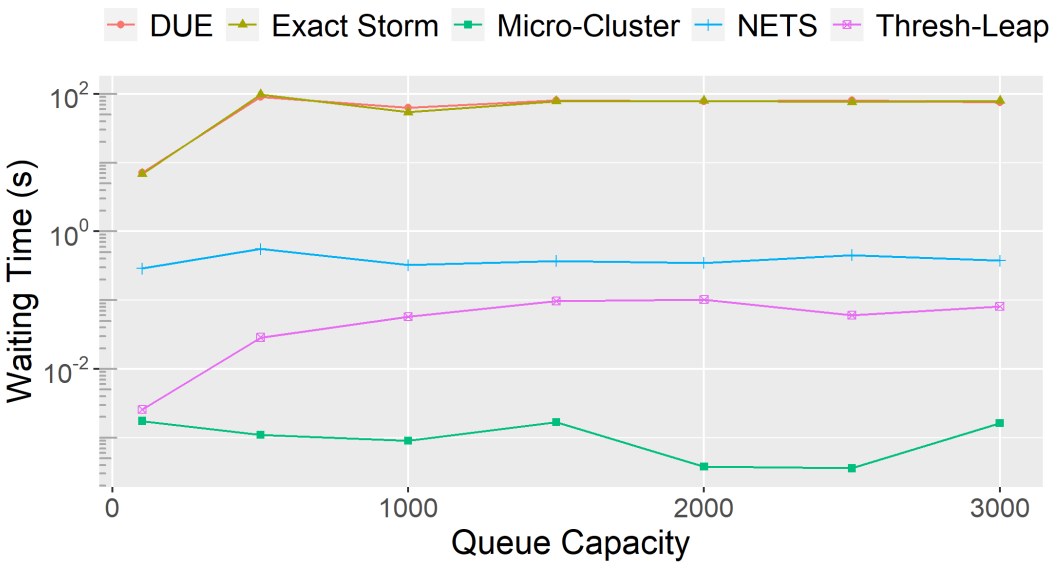

Figure 3.14: Impact of Queue Capacity vs Average Waiting Time for the Stock dataset

Due to a constant average waiting time, there is no significant impact on the total execution time as the queue capacity increases, as can be seen in Figures 3.15, 3.16 and 3.17.

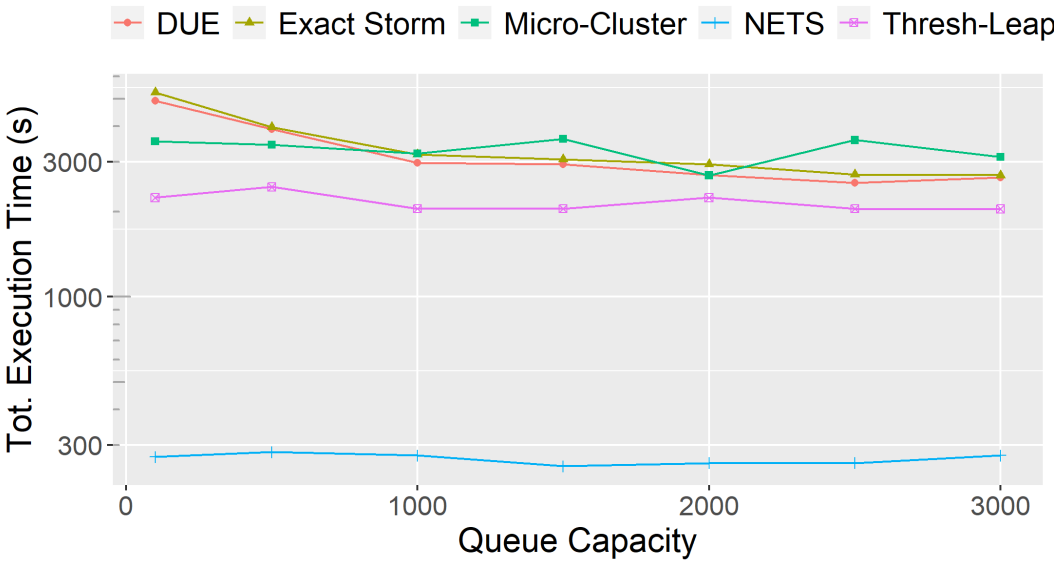

Figure 3.15: Impact of Queue Capacity vs Total Execution Time for the FC dataset

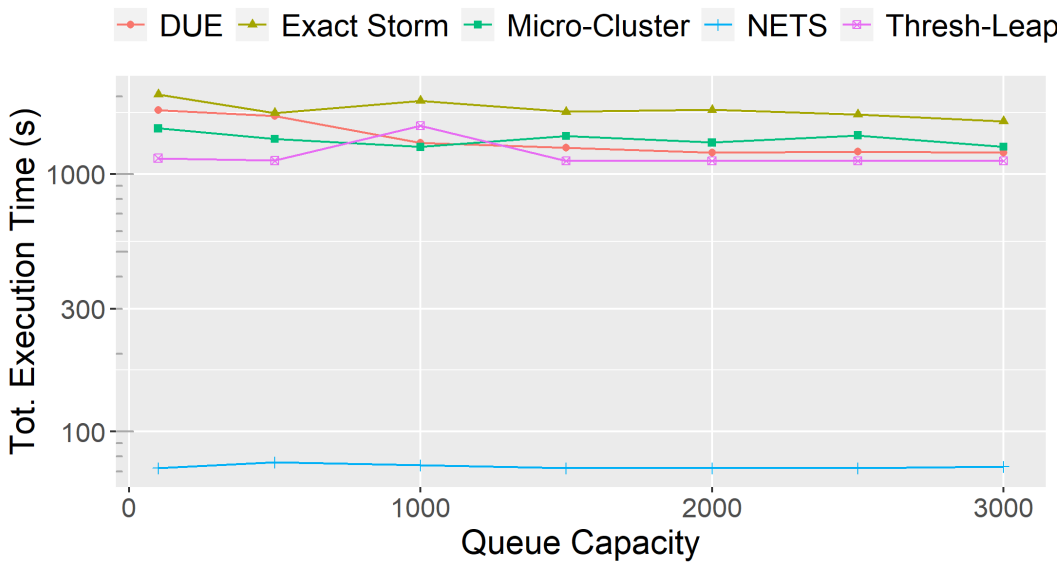

Figure 3.16: Impact of Queue Capacity vs Total Execution Time for the TAO dataset

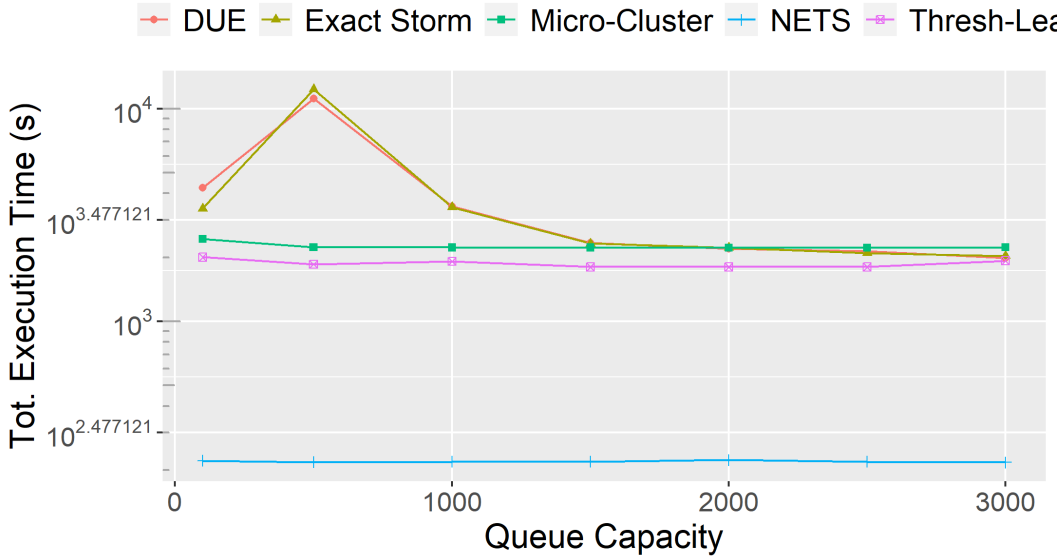

Figure 3.17: Impact of Queue Capacity vs Total Execution Time for the Stock dataset

3.5.4 Impact of Slide Size on Total Time

Now we study the impact of varying slide size on the total execution time and average waiting time.

As the slide size increases, more data points are taken in the window, up to a point where $S = W$, which means that each point takes part in just one window. This is because in this case when the window slides, all the data points in the window are removed. This means that each data point has no preceding neighbor and thus the time that was required to update them is saved [76]. This is contrary to the case where the slide size is less than the window size. In those cases, the expired data points are removed one-by-one.

Initially, when slide size is small, fewer data points are dropped, but as it is increased, more data points are dropped in larger groups, resulting in a decrease in total execution time, as can be observed in Figures 3.18, 3.19 and 3.20. The slide size is increased in the pattern as in the referred paper [76] (in terms of S/W), i.e.: 10 %, 20 %, 50 %, 100 %.

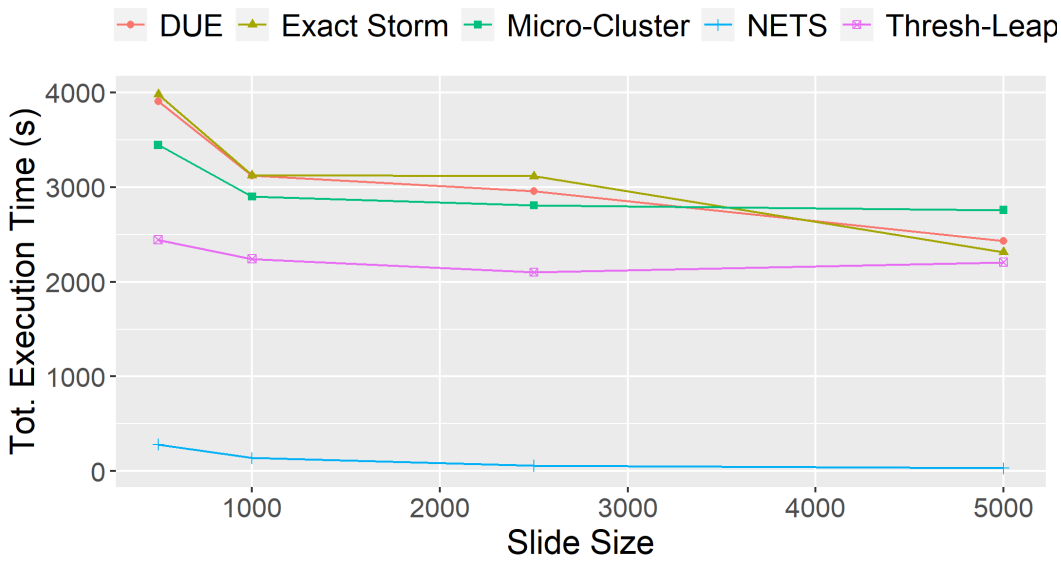

Figure 3.18: Impact of Slide Size vs Total Execution Time for the FC dataset

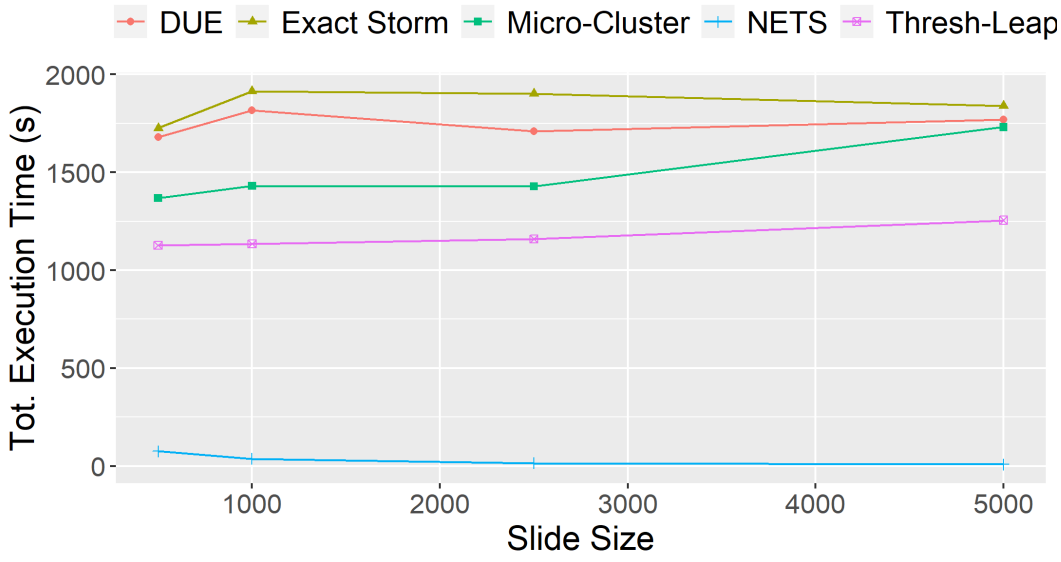

Figure 3.19: Impact of Slide Size vs Total Execution Time for the TAO dataset

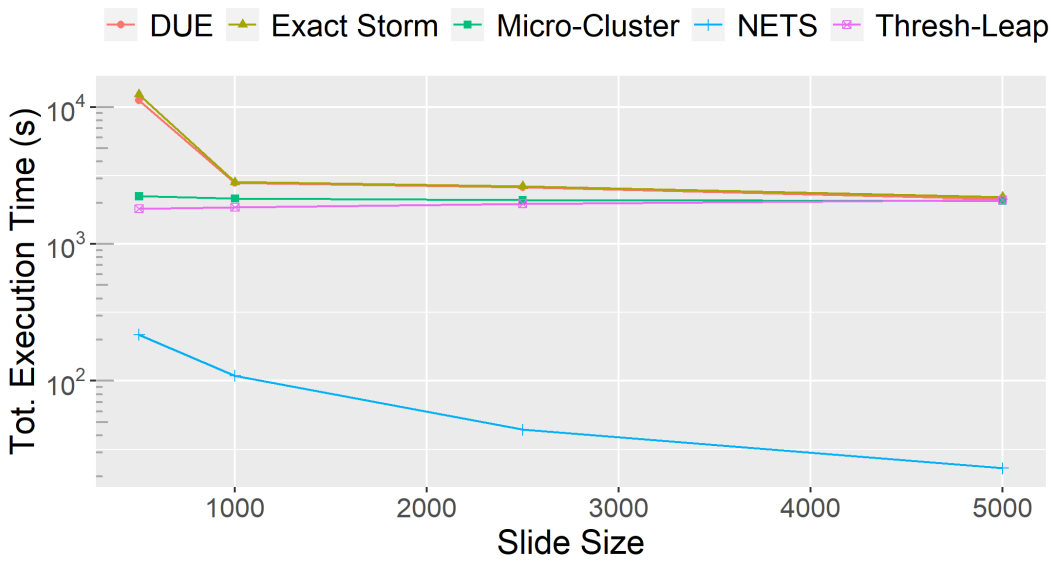

Figure 3.20: Impact of Slide Size vs Total Execution Time for the STOCK dataset

As can be observed in figures 3.21, 3.22 and 3.23, the average waiting times for Exact Storm and DUE decrease with the increase in the slide size, as they continue to drop more data points.

Surprisingly, the performance of NETS gets worse as it starts to lose data points as the slide size increases. This is because as the slide size increases, the algorithm spends more time observing the inter-slide and intra-slide proximity and thus start to lose points rapidly.

On the other hand, Thresh-Leap and Micro-Cluster have more data points in the queue for them to be processed in bulk, thus resulting in an increase in the average waiting time as slide size increases.

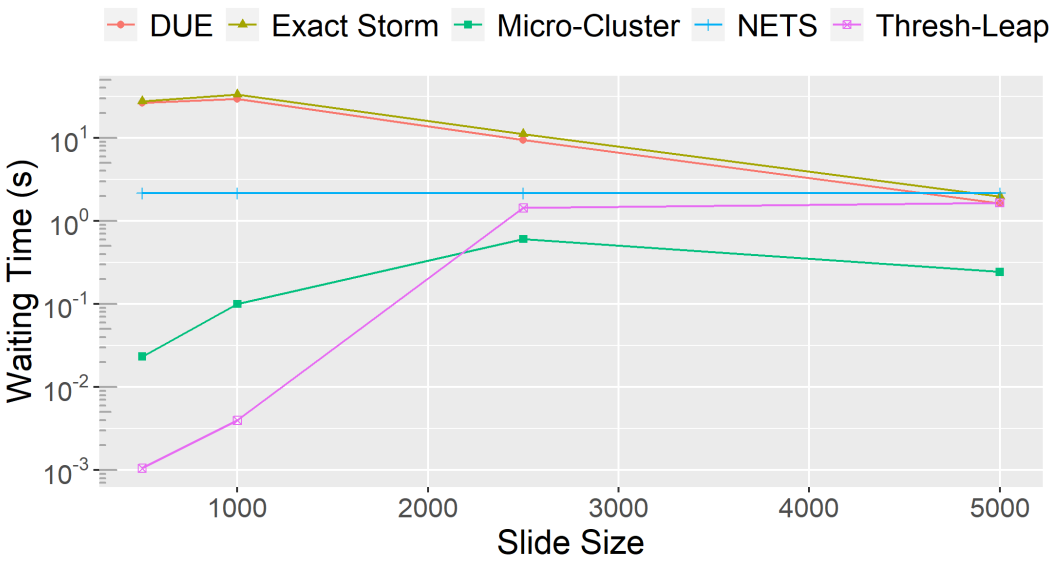

Figure 3.21: Impact of Slide Size vs Average Waiting Time for the FC dataset

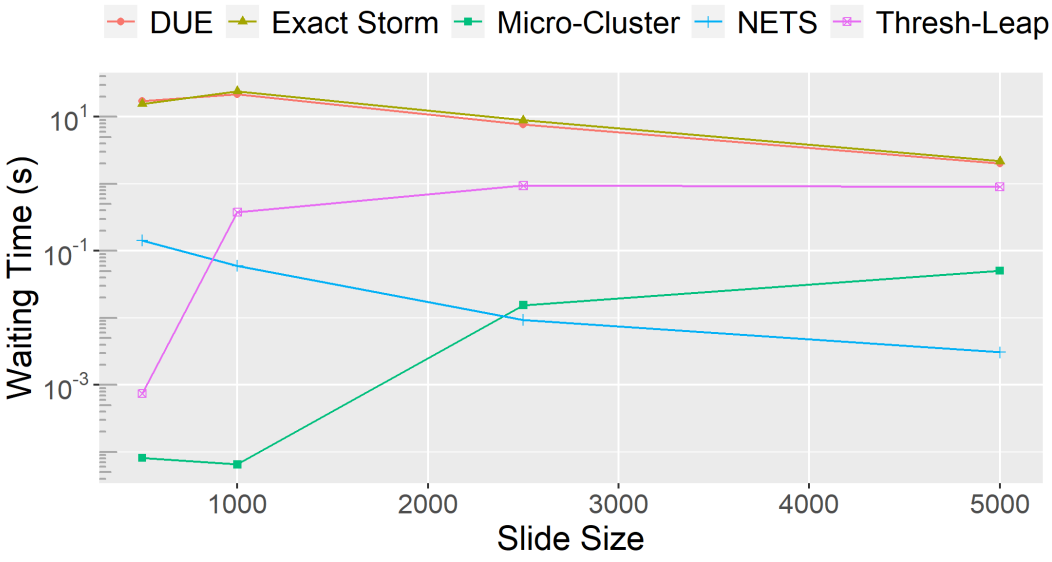

Figure 3.22: Impact of Slide Size vs Average Waiting Time for the TAO dataset

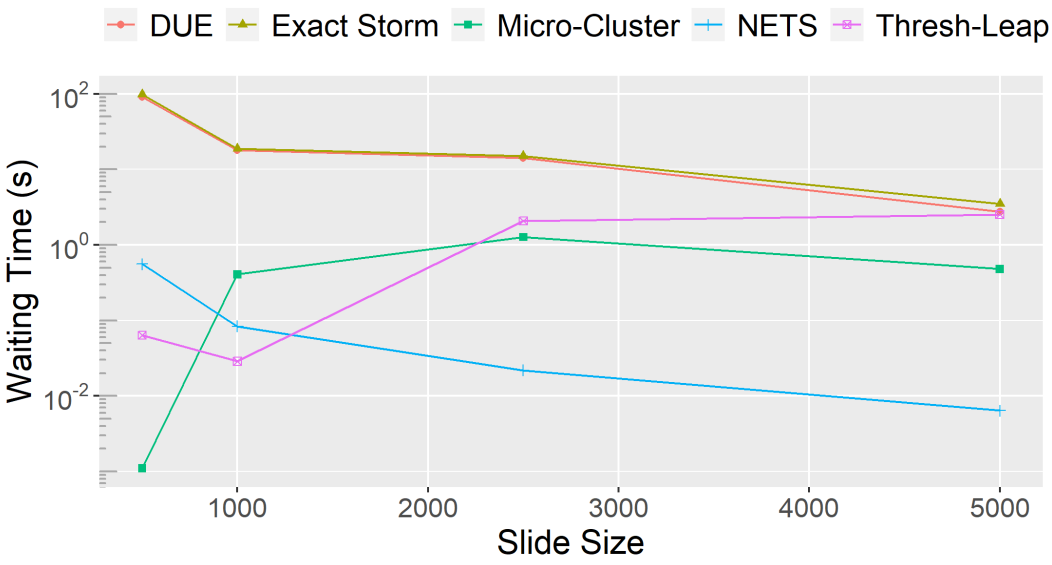

Figure 3.23: Impact of Slide Size vs Average Waiting Time for the STOCK dataset

3.6 Performance Comparison

The performance of each algorithm is studied in terms of precision, recall and F-1 score, obtained at values considered default throughout the experiments. All these values are calculated considering the results produced by the actual algorithms (without varying arrival rate and dropping data points) as ground truth. However, these results do not take into account those data points that were dropped by the algorithms. Table 3.2 represents the results of experiments considering a finite queue capacity and Table 3.3 represents the results of experiments considering infinite queue capacity.

Dataset	Algorithm	Precision	Recall	F-1 Score
Forest Cover	DUE	0.98	0.01	0.01
	Exact-Storm	0.89	0.83	0.86
	Micro-Cluster	1.00	1.00	1.00
	NETS	1.00	1.00	1.00
	Thresh-Leap	1.00	1.00	1.00
TAO	DUE	0.00	0.00	0.00
	Exact-Storm	0.95	0.56	0.70
	Micro-Cluster	0.99	0.99	0.99
	NETS	1.00	1.00	1.00
	Thresh-Leap	1.00	1.00	1.00
STOCK	DUE	0.15	0.01	0.00
	Exact-Storm	0.96	0.31	0.50
	Micro-Cluster	0.99	0.99	0.99
	NETS	1.00	1.00	1.00
	Thresh-Leap	0.99	0.99	0.99

Table 3.2: Performance Comparison of Algorithms Over Different Datasets on Default values assuming Data point dropping over finite Queue capacity

Dataset	Algorithm	Precision	Recall	F-1 Score
Forest Cover	DUE	0.98	0.01	0.01
	Exact-Storm	1.00	1.00	1.00
	Micro-Cluster	1.00	1.00	1.00
	NETS	0.98	1.00	0.99
	Thresh-Leap	1.00	1.00	1.00
TAO	DUE	0.00	0.00	0.00
	Exact-Storm	1.00	1.00	1.00
	Micro-Cluster	1.00	1.00	1.00
	NETS	1.00	1.00	1.00
	Thresh-Leap	1.00	1.00	1.00
STOCK	DUE	0.15	0.00	0.00
	Exact-Storm	1.00	1.00	1.00
	Micro-Cluster	0.99	0.99	0.99
	NETS	1.00	1.00	1.00
	Thresh-Leap	0.99	0.99	0.99

Table 3.3: Performance Comparison of Algorithms Over Different Datasets on Default values assuming Data point dropping over infinite Queue capacity

4 Conclusions and Future Work

In this chapter, we discuss the conclusions in section 4.1 and future work in section 4.2.

4.1 Conclusions

After observing the impact of arrival rate, queue capacity and slide size on total execution time, average waiting time and the total outliers detected out of the actual outliers, we arrive at the following conclusions:

- For all algorithms and assuming that the queue is bounded, the performance of outlier detection algorithms for data streams is dependent on the arrival rate (lambda) of the data points. For example, for FC dataset, Micro-cluster shows approximately 27.9%, Thresh-leap 22.9%, DUE 32.6%, Exact Storm 28.6% and NETS 35.3% faster total execution time when λ=10^6 vs λ=10^3. For TAO dataset, Micro-cluster shows approximately 10.4%, Thresh-leap 9.7%, DUE 15.7%, Exact Storm 15% and NETS 9.7% faster total execution time when λ=10^6 vs λ=10^3. For STOCK dataset, Micro-cluster shows approximately 13.5%, Thresh-leap 9.4%, DUE 49.9%, Exact Storm 58.6% and NETS 8.9% faster total execution time when λ=10^6 vs λ=10^3.

- Assuming that the queue is bounded, as the arrival rate increases, the average waiting time remains constant. This happens for Exact Storm and DUE because

they fail to process the data points fast enough to not let the queue be full. As the data points arrive and find the queue to be full, they are dropped, maintaining the average waiting time to a constant value. For other algorithms, the average time may slightly increase as they struggle avoid losing data points and detect almost all of them at the output.

- Our experiments show that when using a bounded queue for incoming data points and allowing data drop, the arrival rate has a significant detrimental impact on the F-1 score for all algorithms, across all datasets, especially for DUE and Exact Storm. The average detrimental impact of arrival rate, considering a bounded queue and data drop, over 3 datasets, on F-1 score for each algorithm is: 99.78% for Thresh-Leap, 100% for NETS, 99.69% for Micro-Cluster, 67.5% for Exact Storm and 0.422% for DUE.

- The average impact of arrival rate, over 3 datasets, on the number of outliers detected for each algorithm is: 39.2% for Exact Storm, 38.6% for DUE, 0% for Thresh-Leap for NETS, 0% for Micro-Cluster and -0.33%.

- For Exact Storm and DUE, when the queue capacity is small, the average waiting and total execution time is small, because if the queue is full, the data points do not have to wait and are rather dropped. On the other hand, if the queue capacity increases for the two algorithms, the total execution time and average waiting time increase slightly and then become constant because then the queue has enough space to hold the data points and thus no data points are dropped. For other algorithms, as almost data points are lost, their total execution time and average waiting time remain constant with the increase in queue capacity.

- When the slide size is small, the total execution time is comparatively large because fewer data points are taken at a time, increasing the total time to execute them all. It decreases as the slide size increases.

- As the slide size increases, the average waiting time for Exact Storm and DUE decreases as they drop more data points at a time. It remains constant for other algorithms like Micro-cluster and Thresh-leap but the performance of NETS gets worse as it starts to lose more data points as the slide size increases, resulting decrease in the average waiting time, which can be clearly observed in a bigger dataset-STOCK.

- The average waiting time for Exact Storm and DUE algorithms decreases as they drop more data points with the increase in slide size, while those who do not drop data points, observe an increase in the average waiting time. The reason why Exact Storm and DUE perform poorly could be because when the window slides, both algorithms have to go through the stored index structures and $o.pn$ lists for each data point and maintain event queues, resulting in requiring more time to process queries, therefore, when the data arrives at a faster rate, those updates consume the most time, resulting in dropping data points.

4.2 Future Work

In this section, we talk about the future work idea for this research.

- Observe the impact of window size on the performance of each algorithm and choose the optimum window size, using the approach used in [12] or using an adaptive window size for improved estimation [35] and [23].

- Instead of using count-based and time-based sliding window models alone for data streams, experiment implementing the ECM- Sketch technique [57], which allows effective summarization of data streams over both types of sliding window models with accuracy. This can help reduce the time for outlier detection.

- Run the same experiments on density-based [51] [74] [62] [10], angle-based [46] [88] [67] [48] and statistical-approach [85] [16] based outlier detection algorithms to observe their performance in terms of precision, recall and F-1 score.

- Implement similar experiments on clustering-based outlier detection techniques [37, 56, 19, 83] to deal with unsupervised or semi-supervised data and observe the performance in terms of precision, recall and F-1 score, as research in [52] proves that based on their experiments, cluster-based outlier detection techniques are more efficient than the distance and density-based approaches for outlier detection.

- Reduce the dependency of NETS on the increase in slide size, to get a uniform performance through variable parameters.

- Create redundancy at the side of consumer to divide load, resulting in letting each system take data streams as input alternatively and then observe the total number of outliers lost and performance of the system.

- Change the implementation of Producer-Consumer to ED-Tree [1], which is a distributed pool structure combining elimination-tree and diffracting-tree paradigms, which will allow high degree of parallelism or distributed queue [38], rather than through a queue to observe any difference if any.

- Consider the impact of noise and concept drift in the experiments, as done for data stream regression models in [78].

- Run similar experiments for data mining techniques including control chart, linear regression, and Manhattan distance techniques for outlier detection in data mining [11].

- For Exact Storm, instead of storing all the data points in the *o.pn* list, check if it is greater than or equal to K, if it is, do not store it. This can result in reducing the time consumed to go through each data point when the window slides.

- Instead of using the count-based sliding window model over data streams, consider implementing the Selective Repeat ARQ sliding window protocol [84], as used in the Transmission Control Protocol (TCP), where if any frame is corrupted or lost, those selective frames have to be sent again. In this case, selective data points if lost could be sent back again to avoid losing outliers for detection.

References

[1] Yehuda Afek et al. "Scalable producer-consumer pools based on elimination-diffraction trees". In: *European Conference on Parallel Processing*. Springer. 2010, pp. 151–162 (cit. on p. 44).

[2] Charu C Aggarwal. *Data streams: models and algorithms*. Vol. 31. Springer Science & Business Media, 2007 (cit. on p. 1).

[3] Charu C Aggarwal. "Outlier ensembles: position paper". In: *ACM SIGKDD Explorations Newsletter* 14.2 (2013), pp. 49–58 (cit. on p. 1).

[4] Ghadah Aldabbagh, Miguel Rio, and Izzat Darwazeh. "Fair early drop: An active queue management scheme for the control of unresponsive flows". In: *2010 10th IEEE International Conference on Computer and Information Technology*. IEEE. 2010, pp. 2668–2675 (cit. on p. 5).

[5] Kemal Altinkemer, Indranil Bose, and Raktim Pal. "Average waiting time of customers in an M/D/k queue with nonpreemptive priorities". In: *Computers & operations research* 25.4 (1998), pp. 317–328 (cit. on p. 21).

[6] Fabrizio Angiulli and Fabio Fassetti. "Detecting distance-based outliers in streams of data". In: *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*. 2007, pp. 811–820 (cit. on pp. 2, 3, 8, 20).

[7] Fabrizio Angiulli and Clara Pizzuti. “Outlier mining in large high-dimensional data sets”. In: *IEEE transactions on Knowledge and Data engineering* 17.2 (2005), pp. 203–215 (cit. on p. 2).

[8] Brian Babcock et al. “Models and issues in data stream systems”. In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 2002, pp. 1–16 (cit. on p. 2).

[9] Brian Babcock et al. *Sliding window computations over data streams*. Tech. rep. Stanford InfoLab, 2002 (cit. on p. 6).

[10] Mei Bai et al. “An efficient algorithm for distributed density-based outlier detection on big data”. In: *Neurocomputing* 181 (2016), pp. 19–28 (cit. on p. 44).

[11] Zuriana Abu Bakar et al. “A comparative study for outlier detection techniques in data mining”. In: *2006 IEEE conference on cybernetics and intelligent systems*. IEEE. 2006, pp. 1–6 (cit. on p. 45).

[12] Oresti Banos et al. “Window size impact in human activity recognition”. In: *Sensors* 14.4 (2014), pp. 6474–6499 (cit. on p. 43).

[13] Guillermo Barrenetxea et al. “The hitchhiker’s guide to successful wireless sensor network deployments”. In: *Proceedings of the 6th ACM conference on Embedded network sensor systems*. 2008, pp. 43–56 (cit. on p. 1).

[14] Sabyasachi Basu and Martin Meckesheimer. “Automatic outlier detection for time series: an application to sensor data”. In: *Knowledge and Information Systems* 11.2 (2007), pp. 137–154 (cit. on p. 1).

[15] Lei Cao et al. “Scalable distance-based outlier detection over high-volume data streams”. In: *2014 IEEE 30th International Conference on Data Engineering*. IEEE. 2014, pp. 76–87 (cit. on pp. 2, 3, 14, 20).

[16] AC Carrilho, Mauricio Galo, and RC Santos. "STATISTICAL OUTLIER DETECTION METHOD FOR AIRBORNE LIDAR DATA." In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* (2018) (cit. on p. 44).

[17] Ho-Leung Chan et al. "Continuous monitoring of distributed data streams over a time-based sliding window". In: *Algorithmica* 62.3 (2012), pp. 1088–1111 (cit. on p. 6).

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Outlier detection: A survey". In: *ACM Computing Surveys* 14 (2007), p. 15 (cit. on p. 1).

[19] A Christy, G Meera Gandhi, and S Vaithyasubramanian. "Cluster based outlier detection algorithm for healthcare data". In: *Procedia Computer Science* 50 (2015), pp. 209–215 (cit. on p. 44).

[20] RD Clarke. "An application of the Poisson distribution". In: *Journal of the Institute of Actuaries* 72.3 (1946), pp. 481–481 (cit. on p. 18).

[21] Michele Dallachiesa et al. "Sliding windows over uncertain data streams". In: *Knowledge and Information Systems* 45.1 (2015), pp. 159–190 (cit. on p. 19).

[22] Thomas Demoor et al. "Influence of real-time queue capacity on system contents in DiffServ's expedited forwarding per-hop-behavior". In: *Journal of Industrial and Management Optimization* 6.3 (2010), pp. 587–602 (cit. on p. 30).

[23] Mahmood Deypir, Mohammad Hadi Sadreddini, and Sattar Hashemi. "Towards a variable size sliding window model for frequent itemset mining over data streams". In: *Computers & industrial engineering* 63.1 (2012), pp. 161–172 (cit. on p. 43).

[24] Yanlei Diao et al. “Capturing data uncertainty in high-volume stream processing”. In: *arXiv preprint arXiv:0909.1777* (2009) (cit. on p. 2).

[25] Xiaofeng Ding et al. “Continuous monitoring of skylines over uncertain data streams”. In: *Information Sciences* 184.1 (2012), pp. 196–214 (cit. on p. 19).

[26] Paul Dokas et al. “Data mining for network intrusion detection”. In: *Proceeding of NGDM* (Jan. 2) (cit. on p. 1).

[27] Victor Garcia-Font, Carles Garrigues, and Helena Rifà-Pous. “A Comparative Study of Anomaly Detection Techniques for Smart City Wireless Sensor Networks”. In: *Sensors* 16.6 (2016). ISSN: 1424-8220 (cit. on p. 2).

[28] Gebeyehu Belay Gebremeskel et al. “Combined data mining techniques based patient data outlier detection for healthcare safety”. In: *International Journal of Intelligent Computing and Cybernetics* (2016) (cit. on p. 1).

[29] Rainer Gemulla and Wolfgang Lehner. “Sampling time-based sliding windows in bounded space”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008, pp. 379–392 (cit. on p. 19).

[30] Lukasz Golab. “Querying sliding windows over online data streams”. In: *International Conference on Extending Database Technology*. Springer. 2004, pp. 1–11 (cit. on p. 19).

[31] Lukasz Golab, Shaveen Garg, and M Tamer Özsu. “On indexing sliding windows over online data streams”. In: *International Conference on Extending Database Technology*. Springer. 2004, pp. 712–729 (cit. on p. 6).

[32] Cyril Goutte and Eric Gaussier. “A probabilistic interpretation of precision, recall and F-score, with implication for evaluation”. In: *European conference on information retrieval*. Springer. 2005, pp. 345–359 (cit. on p. 21).

[33] Douglas M Hawkins. *Identification of outliers*. Vol. 11. Springer, 1980 (cit. on p. 1).

[34] EventHelix.com Inc. *M/M/1 Queueing System*. URL: http://www.eventhelix.com/RealtimeMantra/congestionControl/m_m_1_queue.htm (visited on 03/10/2021) (cit. on p. 3).

[35] Waheed Iqbal, Josep Lluis Berral, David Carrera, et al. "Adaptive sliding windows for improved estimation of data center resource utilization". In: *Future Generation Computer Systems* 104 (2020), pp. 212–224 (cit. on p. 43).

[36] Kevin Jeffay. "The real-time producer/consumer paradigm: A paradigm for the construction of efficient, predictable real-time systems". In: *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*. 1993, pp. 796–804 (cit. on p. 5).

[37] Sheng-yi Jiang and Qing-bo An. "Clustering-based outlier detection method". In: *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 2. IEEE. 2008, pp. 429–433 (cit. on p. 44).

[38] Theodore Johnson. "Designing a distributed queue". In: *Proceedings. Seventh IEEE Symposium on Parallel and Distributed Processing*. IEEE. 1995, pp. 304–311 (cit. on p. 44).

[39] Carlos Juiz and Ramon Puigjaner. "Improved performance model of a real-time software element: the producer-consumer". In: *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*. IEEE. 1995, pp. 174–178 (cit. on p. 18).

[40] Robert W. Klein and Stephen D. Roberts. "A time-varying Poisson arrival process generator". In: *SIMULATION* 43.4 (1984), pp. 193–195 (cit. on p. 18).

[41] Leonard Kleinrock. “Optimum bribing for queue position”. In: *Operations Research* 15.2 (1967), pp. 304–318 (cit. on p. 21).

[42] Edwin M Knorr and Raymond T Ng. “Algorithms for mining distance-based outliers in large datasets”. In: *VLDB*. Vol. 98. Citeseer. 1998, pp. 392–403 (cit. on p. 1).

[43] Maria Kontaki et al. “Continuous monitoring of distance-based outliers over data streams”. In: *2011 IEEE 27th International Conference on Data Engineering*. IEEE. 2011, pp. 135–146 (cit. on pp. 2, 3, 11, 12, 20).

[44] Laszlo Kozma. “k Nearest Neighbors algorithm (kNN)”. In: *Helsinki University of Technology* (2008) (cit. on p. 13).

[45] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. “Outlier detection techniques”. In: *Tutorial at KDD* 10 (2010), pp. 1–76 (cit. on p. 1).

[46] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. “Angle-based outlier detection in high-dimensional data”. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452 (cit. on p. 44).

[47] Feng Li et al. “Measuring queue capacities of IEEE 802.11 wireless access points”. In: *2007 Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS'07)*. IEEE. 2007, pp. 846–853 (cit. on p. 30).

[48] Xiaojie Li, Jian Cheng Lv, and Dongdong Cheng. “Angle-based outlier detection algorithm with more stable relationships”. In: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1*. Springer. 2015, pp. 433–446 (cit. on p. 44).

[49] Bo Liu et al. "Svdd-based outlier detection on uncertain data". In: *Knowledge and information systems* 34.3 (2013), pp. 597–618 (cit. on p. 1).

[50] Zhang Longbo et al. "A priority random sampling algorithm for time-based sliding windows over weighted streaming data". In: *Proceedings of the 2007 ACM symposium on Applied computing*. 2007, pp. 453–456 (cit. on p. 6).

[51] Mathew X Ma, Henry YT Ngan, and Wei Liu. "Density-based outlier detection by local outlier factor on largescale traffic data". In: *Electronic Imaging* 2016.14 (2016), pp. 1–4 (cit. on p. 44).

[52] Harshada C Mandhare and SR Idate. "A comparative study of cluster based outlier detection, distance based outlier detection and density based outlier detection techniques". In: *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2017, pp. 931–935 (cit. on p. 44).

[53] N.G. Mankiw. *Principles of Economics, 5th edition*. The Introductory-Level Textbook. South-Western Cengage Learning, 2011 (cit. on p. 4).

[54] Raja Muthalagu, Anudeepsekhar Bolimera, and V Kalaichelvi. "Lane detection technique based on perspective transformation and histogram analysis for self-driving cars". In: *Computers & Electrical Engineering* 85 (2020), p. 106653 (cit. on p. 2).

[55] Javad Noorbakhsh et al. "Deep learning-based cross-classifications reveal conserved spatial behaviors within tumor histological images". In: *Nature communications* 11.1 (2020), pp. 1–14 (cit. on p. 5).

[56] Rajendra Pamula, Jatindra Kumar Deka, and Sukumar Nandi. "An outlier detection method based on clustering". In: *2011 Second International Conference*

on Emerging Applications of Information Technology. IEEE. 2011, pp. 253–256 (cit. on p. 44).

[57] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. "Sketch-based querying of distributed sliding-window data streams". In: *arXiv preprint arXiv:1207.0139* (2012) (cit. on p. 44).

[58] Didier Pitrat. "Accidents involving automated control systems in the French industry." In: *Loss Prevention Bulletin* 247 (2016) (cit. on p. 2).

[59] Ali Rahmani et al. "Graph-based approach for outlier detection in sequential data and its application on stock market and weather data". In: *Knowledge-Based Systems* 61 (2014), pp. 89–97 (cit. on p. 1).

[60] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. "Efficient algorithms for mining outliers from large data sets". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 427–438 (cit. on p. 2).

[61] Poonam Rana, Deepika Pahuja, and Ritu Gautam. "A critical review on outlier detection techniques". In: *International Journal of Science and Research* 3.12 (2014), pp. 2394–2403 (cit. on p. 1).

[62] Dongmei Ren, Baoying Wang, and William Perrizo. "Rdf: A density-based outlier detection method using vertical data representation". In: *Fourth IEEE International Conference on Data Mining (ICDM'04)*. IEEE. 2004, pp. 503–506 (cit. on p. 44).

[63] Nicoló Rivetti, Yann Busnel, and Achour Mostefaoui. "Efficiently summarizing data streams over sliding windows". In: *2015 IEEE 14th International Sympo-*

sium on Network Computing and Applications. IEEE. 2015, pp. 151–158 (cit. on p. 6).

[64] Md Shiblee Sadik and Le Gruenwald. "DBOD-DS: Distance based outlier detection for data streams". In: *International Conference on Database and Expert Systems Applications*. Springer. 2010, pp. 122–136 (cit. on pp. 1, 2).

[65] Shiblee Sadik, Le Gruenwald, and Eleazar Leal. "In pursuit of outliers in multi-dimensional data streams". In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE. 2016, pp. 512–521 (cit. on p. 1).

[66] Bo Sheng et al. "Outlier detection in sensor networks". In: *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*. 2007, pp. 219–228 (cit. on p. 2).

[67] Zhaoyu Shou et al. "Outlier detection with enhanced angle-based outlier factor in high-dimensional data stream". In: *Int. J. Innov. Comput. Inf. Control* 14.5 (2018), pp. 1633–1651 (cit. on p. 44).

[68] Meng Shuai et al. "A kalman filter based approach for outlier detection in sensor networks". In: *2008 International Conference on Computer Science and Software Engineering*. Vol. 4. IEEE. 2008, pp. 154–157 (cit. on p. 2).

[69] Karanjit Singh and Shuchita Upadhyaya. "Outlier detection: applications and techniques". In: *International Journal of Computer Science Issues (IJCSI)* 9.1 (2012), p. 307 (cit. on p. 1).

[70] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. "The 8 requirements of real-time stream processing". In: *ACM Sigmod Record* 34.4 (2005), pp. 42–47 (cit. on p. 1).

[71] Sharmila Subramaniam et al. “Online outlier detection in sensor data using non-parametric models”. In: *Proceedings of the 32nd international conference on Very large data bases*. 2006, pp. 187–198 (cit. on p. 2).

[72] Mahito Sugiyama. “Distance-Based Outlier Detection via Sampling”. In: (2014) (cit. on p. 1).

[73] NNR Murty Ranga Suri, M Narasimha Murty, and G Athithan. *Outlier detection: techniques and applications*. Springer, 2019 (cit. on p. 1).

[74] Bo Tang and Haibo He. “A local density-based approach for outlier detection”. In: *Neurocomputing* 241 (2017), pp. 171–180 (cit. on p. 44).

[75] M. Templ, J. Gussenbauer, and P. Filzmoser. “Evaluation of robust outlier detection methods for zero-inflated complex data”. In: *Journal of Applied Statistics* 47.7 (2020), pp. 1144–1167 (cit. on p. 2).

[76] Luan Tran, Liyue Fan, and Cyrus Shahabi. “Distance-based Outlier Detection in Data Streams”. In: *Proc. VLDB Endow.* 9.12 (Aug. 2016), pp. 1089–1100. ISSN: 2150-8097 (cit. on pp. 8, 20, 27, 34).

[77] Thanh TL Tran et al. “CLARO: modeling and processing uncertain data streams”. In: *The VLDB Journal* 21.5 (2012), pp. 651–676 (cit. on p. 2).

[78] Katharina Tschumitschew and Frank Klawonn. “Effects of drift and noise on the optimal sliding window size for data stream regression models”. In: *Communications in Statistics-Theory and Methods* 46.10 (2017), pp. 5109–5132 (cit. on p. 44).

[79] VM Van Zoest, A Stein, and G Hoek. “Outlier detection in urban air quality sensor networks”. In: *Water, Air, & Soil Pollution* 229.4 (2018), pp. 1–13 (cit. on p. 2).

[80] Jordi Vilaplana et al. “A queuing theory model for cloud computing”. In: *The Journal of Supercomputing* 69.1 (2014), pp. 492–507 (cit. on p. 22).

[81] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. “Progress in outlier detection techniques: A survey”. In: *IEEE Access* 7 (2019), pp. 107964–108000 (cit. on p. 1).

[82] Liangzhou Wang and Chaobin Wang. “Producer-consumer Model Based Thread Pool Design”. In: *Journal of Physics: Conference Series* 1616 (Aug. 2020), p. 012073 (cit. on p. 5).

[83] Xiaochun Wang, Xiali Wang, and Mitch Wilkes. “A k-Nearest Neighbour Spectral Clustering-Based Outlier Detection Technique”. In: *New Developments in Unsupervised Outlier Detection*. Springer, 2021, pp. 147–172 (cit. on p. 44).

[84] El Weldon. “An improved selective-repeat ARQ strategy”. In: *IEEE Transactions on Communications* 30.3 (1982), pp. 480–486 (cit. on p. 45).

[85] James M Whitacre, Tuan Q Pham, and Ruhul A Sarker. “Use of statistical outlier detection method in adaptive evolutionary algorithms”. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 1345–1352 (cit. on p. 44).

[86] Di Yang, Elke A Rundensteiner, and Matthew O Ward. “Neighbor-based pattern detection for windows over streaming data”. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. 2009, pp. 529–540 (cit. on pp. 2, 3, 9, 20).

[87] Ou Yang and Wendi Heinzelman. “Modeling and throughput analysis for SMAC with a finite queue capacity”. In: *2009 International Conference on Intelligent*

Sensors, Sensor Networks and Information Processing (ISSNIP). IEEE. 2009, pp. 409–414 (cit. on p. 19).

[88] Hao Ye, Hiroyuki Kitagawa, and Jun Xiao. "Continuous angle-based outlier detection on high-dimensional data streams". In: *Proceedings of the 19th International Database Engineering & Applications Symposium*. 2015, pp. 162–167 (cit. on p. 44).

[89] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. "NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing". In: *Proceedings of the VLDB Endowment* 12.11 (2019), pp. 1303–1315 (cit. on pp. 2–5, 16, 20).

[90] Longbo Zhang et al. "Random sampling algorithms for sliding windows over data streams". In: *Proceedings Of The 11Th Joint International Computer Conference: JICC 2005*. World Scientific. 2005, pp. 572–575 (cit. on pp. 6, 19).